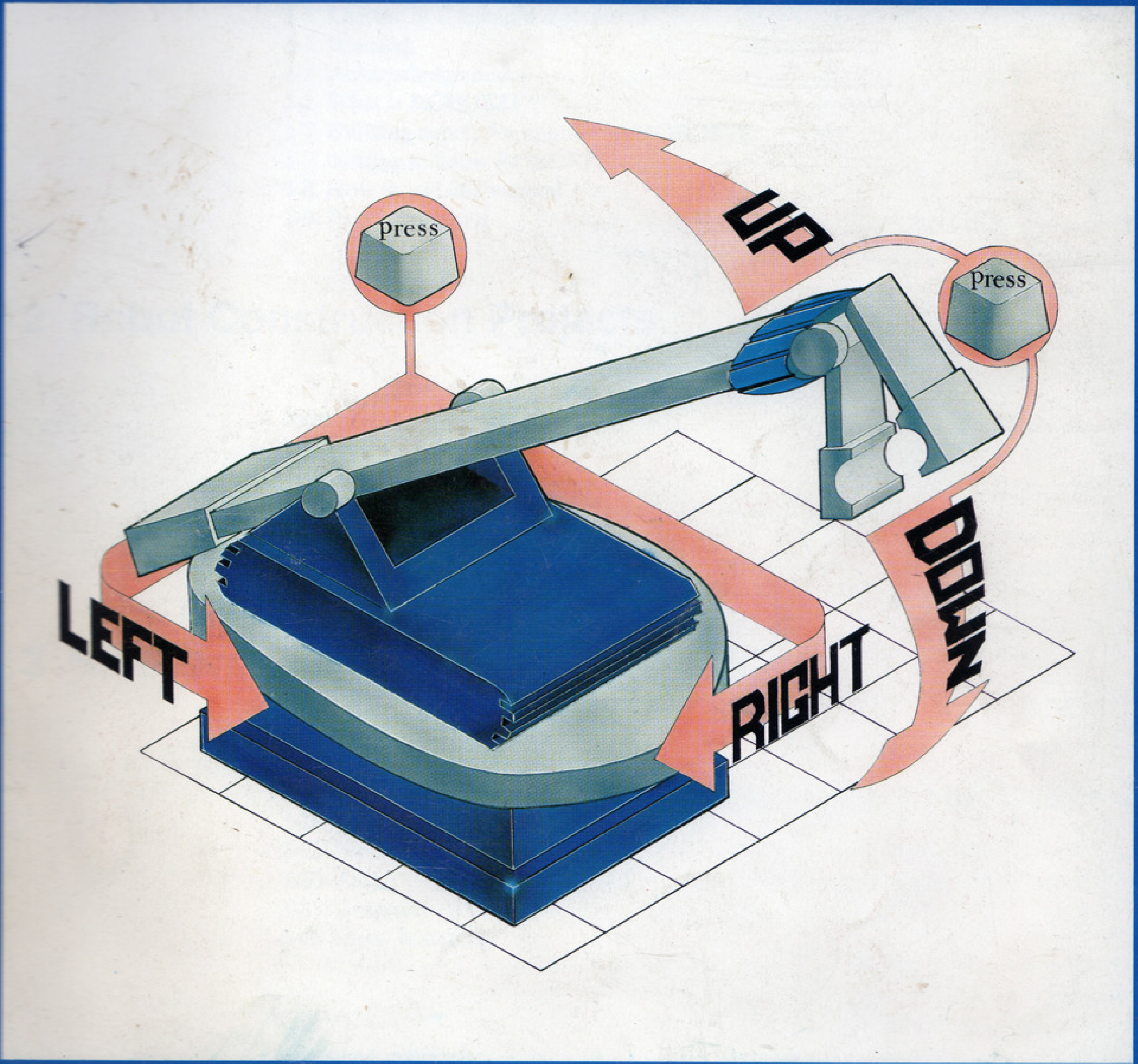
ATARI®

ROBOKIT


press
press
UP
DOWN
LEFT
RIGHT

THE ROBOTICS PACKAGE FOR ALL
ATARI ST COMPUTER SYSTEMS

0. Table of Contents

1 Introduction to ROBOKIT

Page . . .

1-1 License & Copyright statements
1-1 Warning
1-1 Acknowledgements
1-2 What is ROBOKIT ?
1-7 Building robots for control by ROBOKIT
1-7 Getting to know ROBOKIT
1-8 How to use this manual
1-9 Further projects

2 Robot Construction Projects

Page . . .

2-3 Mini Arm
2-11 Lift Operator
2-16 Card Reader
2-21 Drum Plotter
2-28 Maxi-arm

3 Software Guide

Page . . .

3-2 Getting started
3-16 Sequences
3-25 Menus - reference guide
3-30 Text
3-35 Graphics
3-45 Input Watching

4 Robokit Slot Reference Section

Page . . .
4-2 Slot Name : OUTPUT
4-3 Slot Name : INPUT
4-5 Slot Name : MOTORS.
4-10 Slot Name : SERVOS
4-12 Slot Name: BUGGY
4-17 Slot Name : WANDER
4-18 Slot Name : PLAYNAME
4-19 Slot Name : PLAY
4-19 Slot Name : CRDREAD
4-21 Slot Name : IMAGES
4-22 Slot Name : TURTLE
4-25 Slot Name : GRAPHICS
4-27 Slot Name : XYPLOT.
4-30 Slot Name : PRINT-TEXT

5 Software reference section

Page . . .
5-2 PROS Operating System, Slots and Drivers
5-4 File system structures
5-5 MAKEDRP.PRG
5-6 Accessing the ROBOKIT interface card

6 Hardware Interface Reference Section

Page . . .
6-2 Board functions
6-3 Options
6-4 Power up sequence
6-4 Driving motors
6-5 The Prototyping Area
6-7 WARNINGS & ABSOLUTE MAXIMUM RATINGS
6-8 User upgrades and options

1. Introduction to ROBOKIT

1.1. License & Copyright statements

© 1989 by Personal Robots Limited

The ROBOKIT software program, the layout of the Interface Card, and this User G are copyrighted by Personal Robots Limited.

Atari Corporation has license to sell and distribute the software, Interface Card, User Guide. The purchase price of the software disks includes a personal non-tran rable non-exclusive sub-license for the individual/firm to use ROBOKIT software terface Card and User Guide for their own purposes on one computer at any one t

This sub-license does NOT include the right to make copies of the ROBOKIT soft Interface Card or User Guide for resale or transfer without the express permissio the copyright owners.

The user undertakes not to change, modify, add to or delete any of the software co ROBOKIT.

1.2. Warning

Robokit is designed to operate battery driven robot models from an Atari ST through a ROBOKIT ST interface only. Robokit has been designed to be used by people with no previous experience of electronics or robotic control. However, failure to correctly follow the wiring instructions in this manual, could lead to damaging the ROBOKIT ST interface, or your computer.

1.3. Acknowledgements

Robokit is a trademark of Personal Robots Limited. Atari is a trademark of Atari C poration, Inc. LEGO® is a trademark of the LEGO Group. Gem is a trademark of I tal Research Corporation.

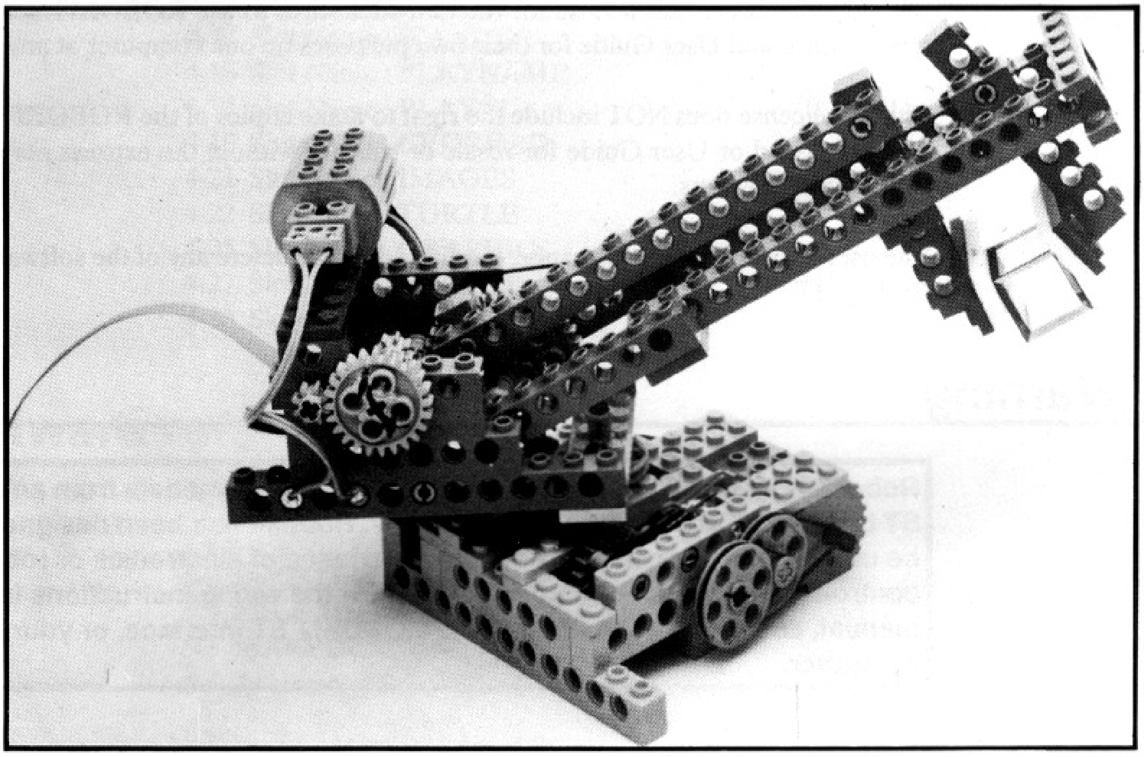
The practical projects in this manual were devised by William Clark and origin published in a series of books called 'Make and program your own robots' (publis by Hutchinson - Beaver Books). The illustrations and instructional text for these proj is the copyright of Lionheart books.

The Software Guide in this manual (Chapter 3) was written by Brian Morris. Other tions were contributed by the Personal Robots Limited team, including: Richard B Andrew Hobson, Richard Pawson, Doug Cunningham and Gary Lawman.

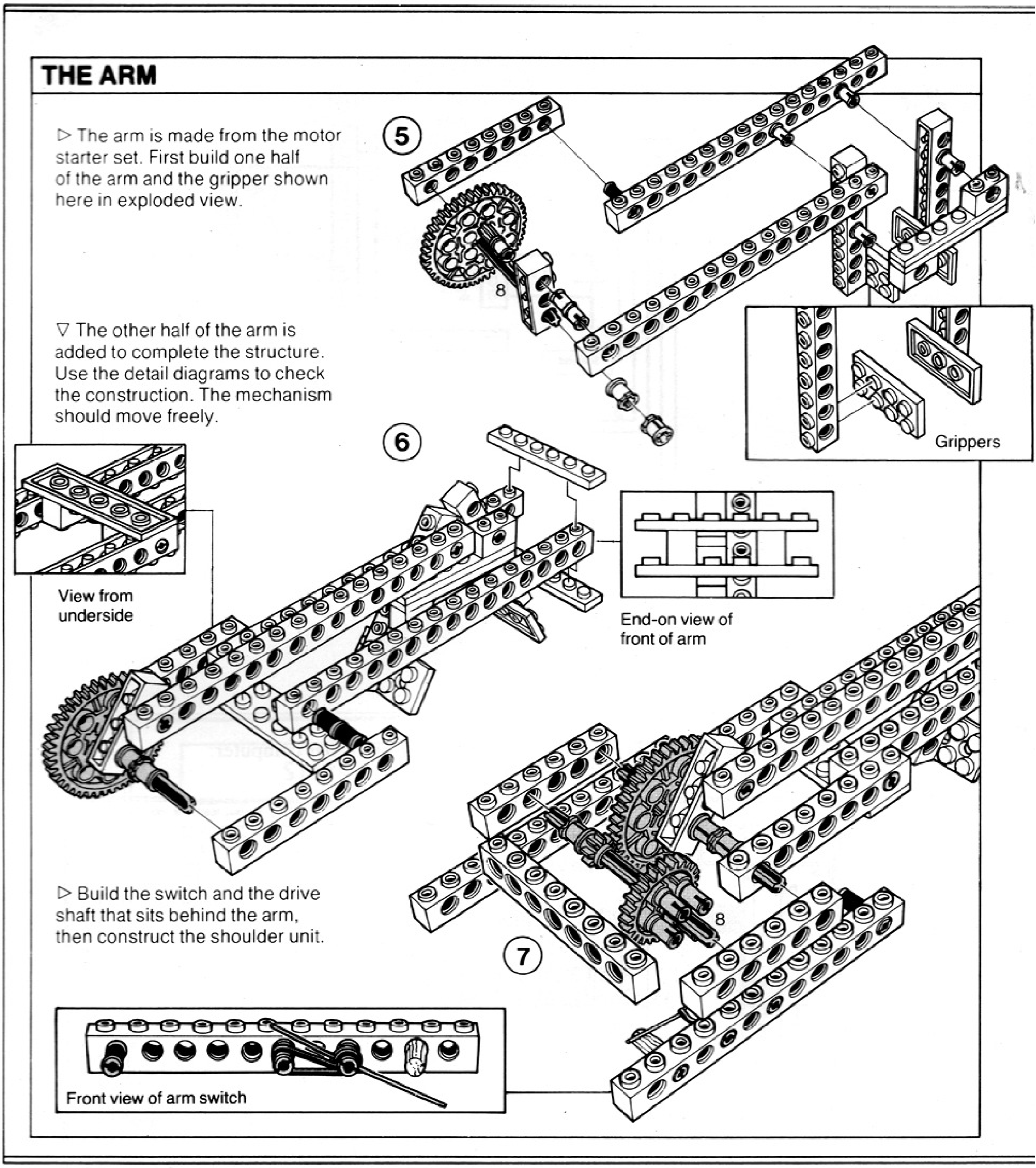
1.4. What is ROBOKIT ?

Atari ROBOKIT is a powerful software package that will enable you to control a wide variety of robot models.

The ROBOKIT manual will show you how to build your own robot models like this:



With step by step instructions like this:



THE ARM

▷ The arm is made from the motor starter set. First build one half of the arm and the gripper shown here in exploded view.

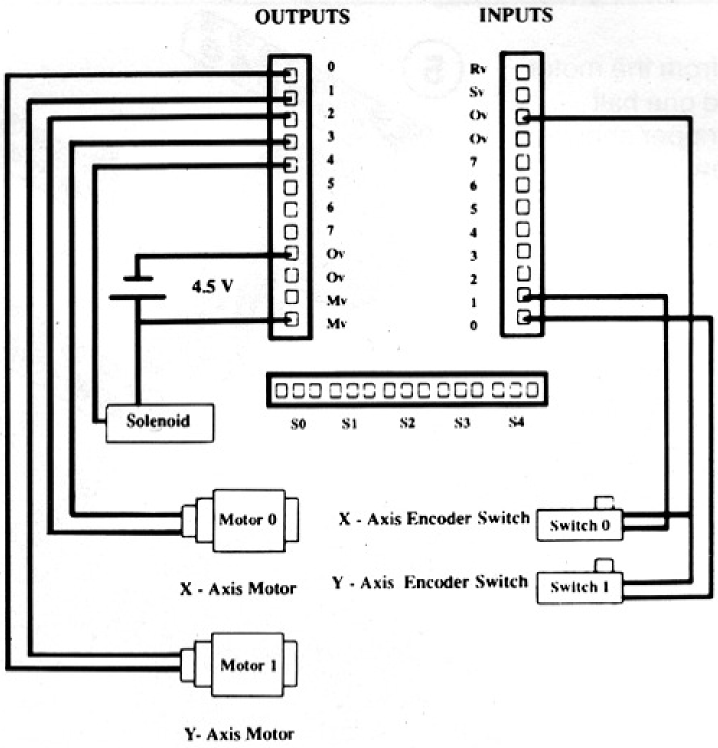
▽ The other half of the arm is added to complete the structure. Use the detail diagrams to check the construction. The mechanism should move freely.

▷ Build the switch and the drive shaft that sits behind the arm, then construct the shoulder unit.

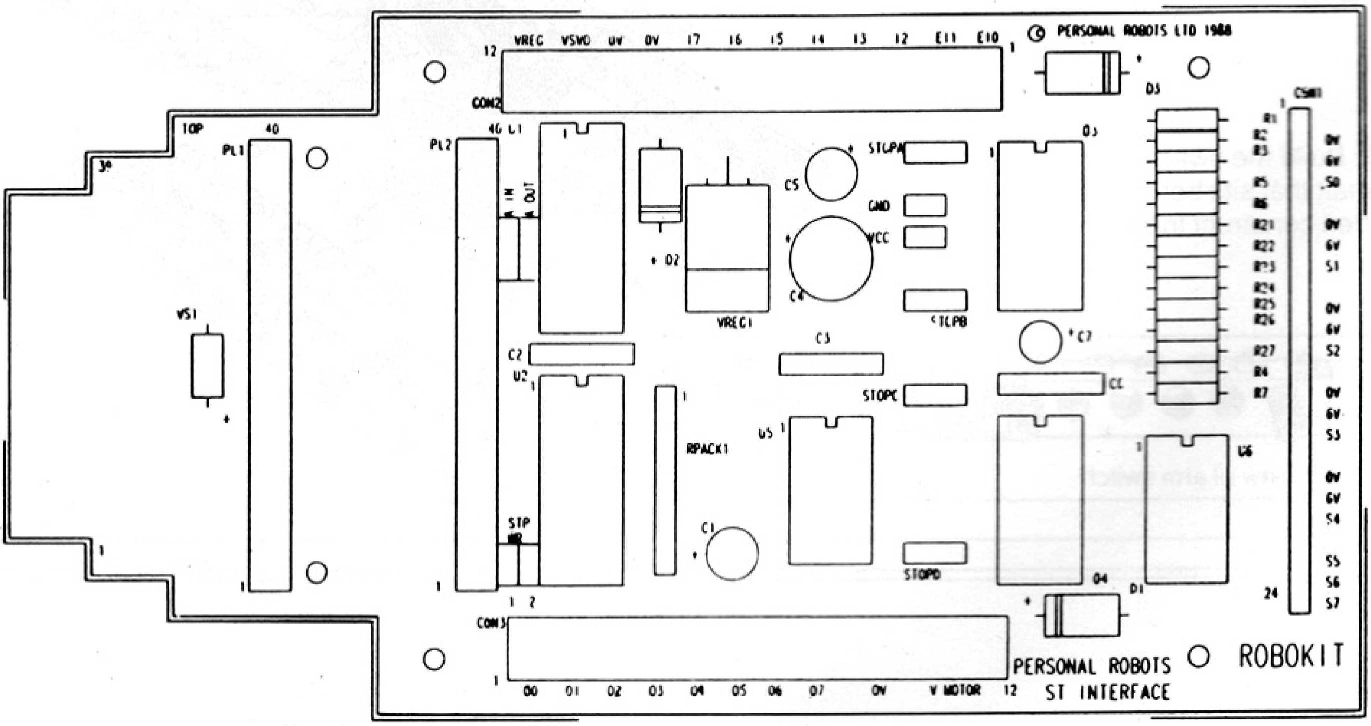
5
8
Grippers
6
View from underside
End-on view of front of arm
7
8
Front view of arm switch

Once you have built the model, there are instructions for wiring up the electrical components like this:

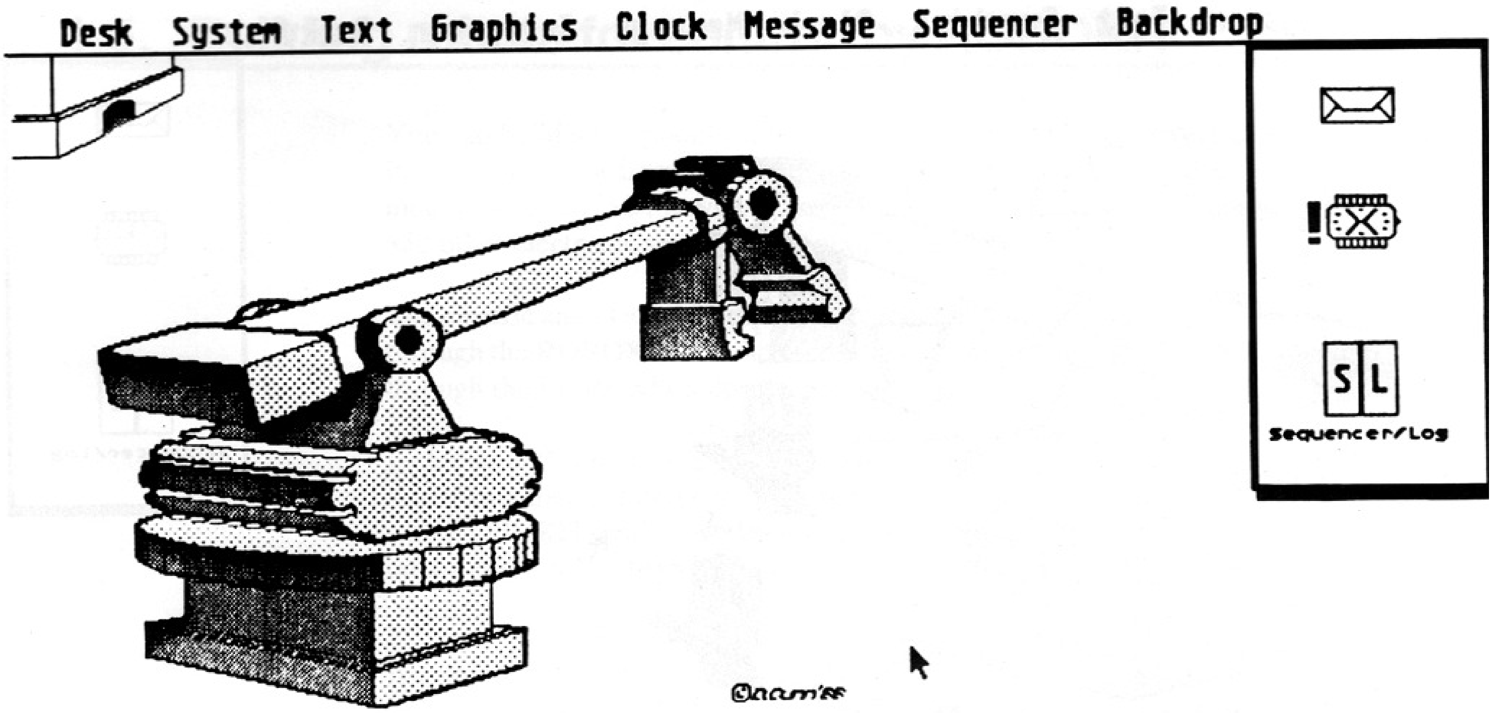
X–Y Plotter Wiring Diagram


OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
Ov
Ov
Mv
Mv
Rv
Sv
Ov
Ov
7
6
5
4
3
2
1
0
4.5 V
Solenoid
S0
S1
S2
S3
S4
Motor 0
X - Axis Encoder Switch
Switch 0
X - Axis Motor
Y - Axis Encoder Switch
Switch 1
Motor 1
Y- Axis Motor

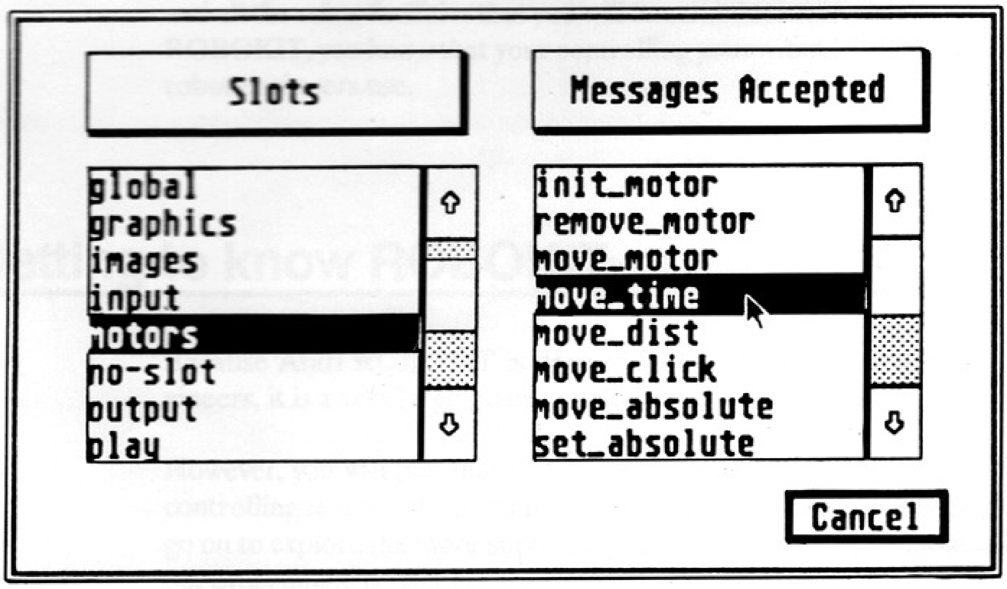
to the ROBOKIT ST interface, which plugs into your computer:


© PERSONAL ROBOTS LTD 1988
VREG VSVO UV OV I7 I6 I5 I4 I3 I2 EI1 EI0
CON2
CON3
TOP
PL1
PL2
VS1
C1
C2
C3
C4
C5
C7
U2
U5
U6
D1
D2
D3
D4
VREG1
RPACK1
STOPA
STOPB
STOPC
STOPD
GND
VCC
CSW1
00 01 02 03 04 05 06 07 0V V MOTOR
0V
6V
S0
S1
S2
S3
S4
S5
S6
S7
PERSONAL ROBOTS
ST INTERFACE
ROBOKIT

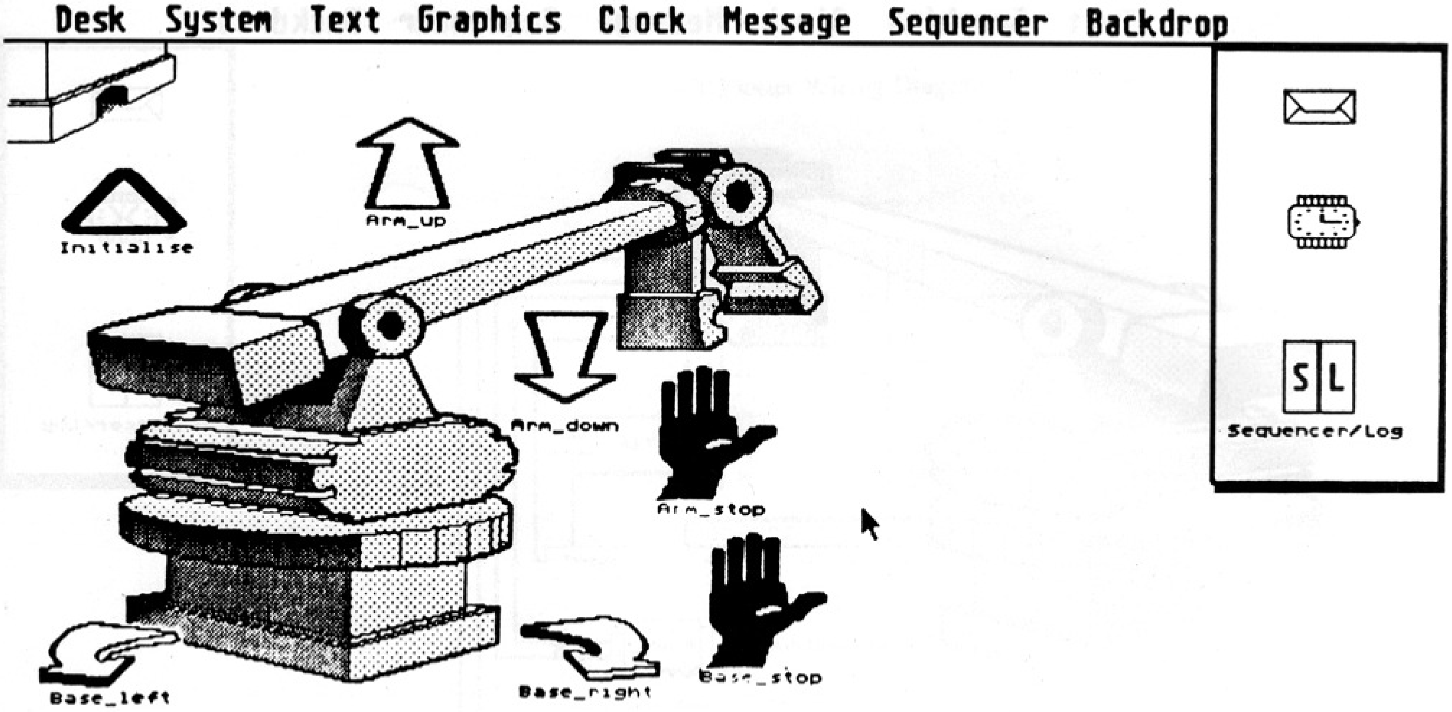
Then you will be shown how to load the ROBOKIT software together with a graphic picture of the robot you have just built:


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
S L
Sequencer/Log

You will be able to send messages like this to any of the motors on the model to control movement:


Slots
Messages Accepted
global
graphics
images
input
motors
no-slot
output
play
init_motor
remove_motor
move_motor
move_time
move_dist
move_click
move_absolute
set_absolute
Cancel

Then you can attach messages to icons on the picture - so that you can control the robot merely by pointing with the mouse to various arrows and symbols:


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
Initialise
Arm_up
Arm_down
Arm_stop
Base_left
Base_right
Base_stop
S L
Sequencer/Log

From this friendly user interface, you can go on to to develop complex applications, including remembering and playing back sequences of moves, and programs that deal with unexpected events.

1.5. Building robots for control by ROBOKIT

You can build such model robots from Lego, from other construction systems such as Fischertechnik, or from modelling materials such as plastic, metal and balsa wood. The models can contain battery powered motors, electromagnets or solenoids, microswitches and other electronic sensors for sound, light, touch or heat.

The electric and electronic parts of any robot model can be connected to your Atari ST through the ROBOKIT ST interface. Once connected, all the control can be performed through the ROBOKIT software package.

Included in this manual are some suggestions for robot models that you can build from a small number of Lego parts. These models show you the kind of projects you can do with ROBOKIT, and show you how to get to use the software. But don't stop there: go on to design and build your own robot models, and bring them to life through ROBOKIT.

Because ROBOKIT can monitor feedback sensors as well as driving motors, ROBOKIT can control real robots - not just dumb robot toys. In fact, using ROBOKIT is the best possible way to learn about robotics - the science of robots. So build as many sensors as you can into your robots. Lego have recently introduced shaft encoders and microswitches packaged in standard Lego bricks - and they operate just like scaled down versions of the devices used on industrial robots. These new devices form part of the schools range, so they're not yet available in the shops unfortunately.

Atari ROBOKIT is actually a low-priced version of a package called Professional ROBOKIT - which powers industrial robots and all kinds of research robots. Professional ROBOKIT uses all the same techniques (slots, messages, windows, system clock and all the other facilities that you will learn about in this manual) so when you use Atari ROBOKIT, you know that your controlling your robot in the same way that professional robot engineers use.

1.6. Getting to know ROBOKIT

Because Atari ROBOKIT is derived from a package used by professional robot engineers, it is a very large piece of software and contains hundreds of functions.

However, you will find that ROBOKIT is very easy to use, and you will be able to start controlling real robots in a matter of minutes. Once you've mastered the basics, you can go on to explore the more sophisticated functions: it may be months before you've tried out every single possibility.

This manual is divided into several sections. The first section after this introduction (Construction Projects) shows how to build five robot models using Lego, and how to wire them up to the ROBOKIT interface, which is plugged into the ROM cartridge socket of your Atari ST.

The next section (Software Guide) explains the operation of all the software features in ROBOKIT, starting with the simple ones and going on to the really advanced functions.

The Reference Section provides more technical information, to which you can refer once you have been though the Software Guide. It contains an explanation of the operating system on which ROBOKIT is based (PROSE - Personal Robot Operating System & Environment), a guide to the various routines (slots) that provide robot functions, a guide to using the ROBOKIT interface, explanation of the image conversion utility (for users with the Neochrome and Dagar packages), and some notes concerning disk and file management for when you start to create your own applications.

1.7. How to use this manual

The ROBOKIT package will run satisfactorily without being connected to a robot model or the interface. You can learn how to use the package, and you can experiment with the Turtle Graphics functions in this way.

But the real fun comes when you control a real robot model. So we suggest that you start by building the Mini-arm (a scaled down version of an industrial robot manipulator) which is the first project in the Projects section. There are instructions for wiring up the model to the ROBOKIT ST Interface.

Important: If you are not familiar with wiring up electrical circuits to a computer interface, and with the necessary precautions to prevent damage, then first read the Interface Guide in the Reference section of this manual.

Once you have the robot wired up to the interface, you can proceed to load and run the ROBOKIT software. The first section of the software guide is called Getting Started, and this will explain everything that you have to do, step by step.

1.8. Further projects

Once you have built and controlled each of the models suggested in this manual, you will naturally want to go on to designing and building your own robot models. Start by modifying the models we've suggested: with additional motors and sensors, or using a different design for the structure.

If you would like some additional project ideas, Lego supply some excellent designs with their 1090 and 1092 Technic Control sets. (These sets are part of the schools range).The motors and shaft encoders are fully compatible with the ROBOKIT interface, so there is no need to use Lego's own interfaces. These sets are not available in the shops, but your school may have them. Both sets can be purchased from: Commotion, 241 Green St. Enfield EN3 7TD, UK, and from other educational suppliers. Commotion also supply a wide range of robotic components, construction kit items and modelling materials. Their catalogue makes good reading for any budding robot engineer.

'The Robot Book', by Richard Pawson (published by Windward) contains details of a dozen robot projects that can be built from Lego and Fishertechnik, complete with step by step diagrams. The price is 7.95 paperback and 12.95 hardback - order it from your local bookshop or from an educational supplier like Commotion. Most of the models are suitable for linking to ROBOKIT. 'Robot Projects' by the same author and publisher is a cheaper publication containing a few of the same projects.

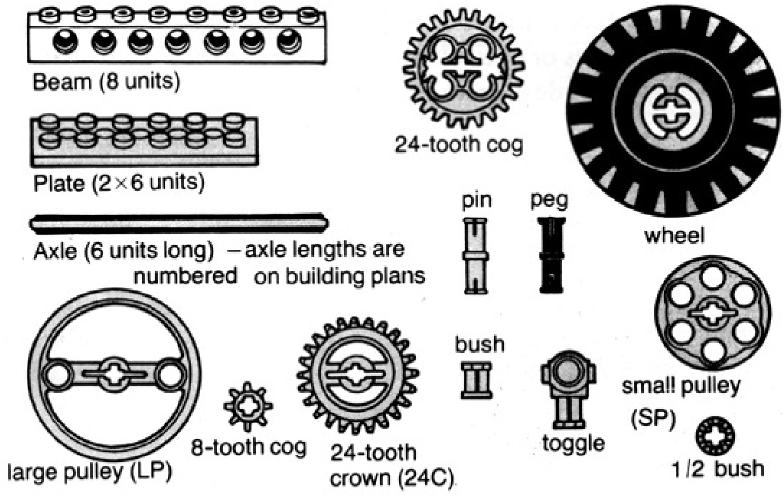
2. Robot Construction Projects

This section of the manual shows you how to build five robotic projects using LEGO Technic - all suitable for control via Atari Robokit. We suggest that you start by building the first project - the Mini-Arm - using the application notes to help you connect the model to the interface. Chapter 3 - the Tutorial Guide to Robokit then shows you how to drive this model.

Once you have built the Mini-Arm, you can go on to try the other four construction projects, using the same principles to connect up the motors and microswitches, and to drive the model from your computer. However, by that stage you may already be itching to design and build your own robot models - from LEGO or from other building materials. This is the whole idea of Robokit.

To help you construct the models, we have included step-by-step illustrations. The Key, below, shows the labelling system that we have adopted to represent the different LEGO parts.

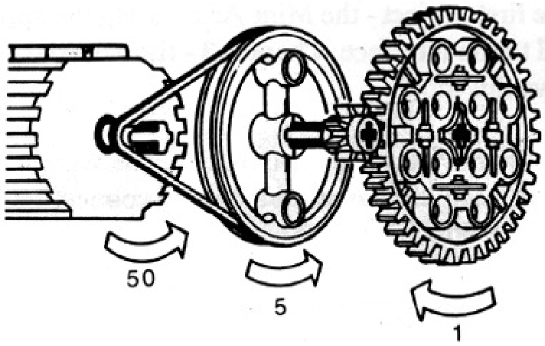
Figure 2.1 Key to labelling of LEGO parts


Beam (8 units)
24-tooth cog
Plate (2×6 units)
pin
peg
wheel
Axle (6 units long) – axle lengths are numbered on building plans
bush
small pulley (SP)
toggle
1/2 bush
8-tooth cog
large pulley (LP)
24-tooth crown (24C)

Most robot models are simple to control provided that the moving parts operate slowly. This means that the standard LEGO motors must be geared down quite considerably. This can be done using the ready-made LEGO gearboxes, through individual gears or pulleys, or a combination of all three. We found it convenient to take the initial power from the motor using a belt to drive a large pulley wheel (LP). This gears the speed down by a factor of 10. Next by using an 8-tooth cog to drive a 40-tooth cog, a further reduction of 5 to 1 is achieved. The two stages of gearing used together will therefore provide a total reduction of 50 to 1. By using different combinations of gear wheels and pulleys, it is possible to achieve a wide range of reduction ratios.

Note that belt & pulley drives can slip when driving a load. This may be useful from a safety viewpoint (it helps to prevent motor damage if the mechanism becomes jammed) - but it can reduce the accuracy of the robots movements. When you come to using fine position control it is therefore a good idea to replace belt drives with gear mechanisms.

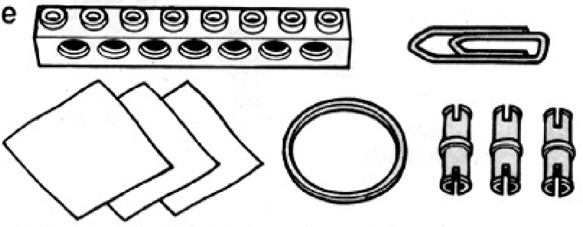
Figure 2.2 Gearing effect of pulleys and gearwheels



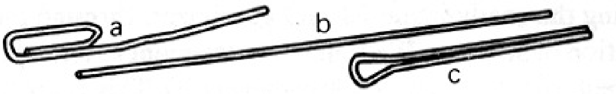
If robots are to operate accurately, they require some form of feedback - to advise the controlling computer whether the robot has moved to where it should be. Some of the models in this manual provide feedback though one or more microswitches. If you can get access to the LEGO Control range of components, then you can use their microswitches or optical shaft encoders to provide an excellent form of feedback - both these types of devices can be driven from the Robokit interface.

The projects shown in this chapter make use of one or more home-made microswitches constructed from LEGO and household materials. The instructions for building the microswitch are shown below.

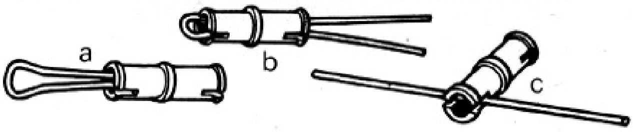
Figure 2.3 Instructions for building the microswitch



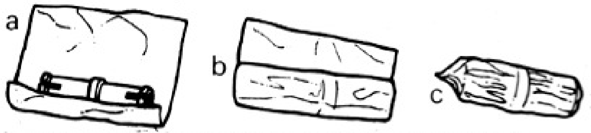
1 Straighten out the paper clip with your fingers or, better still, with pliers.



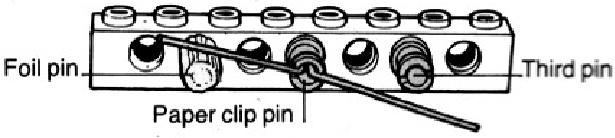
2 Bend the loop into the shape shown then poke it through a pin with a hole through it. The loop at the end of the clip should be about 3mm across.



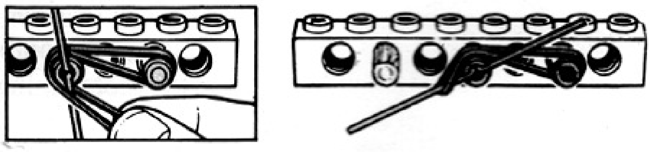
3 Bend back the ends of the clip outwards and back and through the slots in the pin.



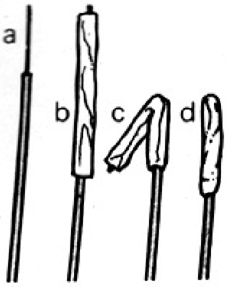
4 Wrap another pin with a square of foil. Then push the three pins into the holes indicated.



5 Hook the belt onto the third pin. Pass it behind the paper clip, pull it down round the middle pin, then pass the loop of the belt over the lower end of the clip. (See step 8, right, for final view.)



6 Make the plugs. Strip one end of each of the two wires by 1cm. Wrap a square of foil round the end of each wire so that 2mm of bare wire poke out. Fold the foil over and roll the end of each wire to form a short plug. Attach the switch to the model and fit the plugs.



2.1. Mini Arm

For our first robotic construction project, we shall build the mini-arm pictured below, using the list of Lego parts shown on the right. You will need to build two of the paper-clip microswitches that were explained at the start of this chapter, though if you have access to the Lego shaft encoders you can adapt the design to take those - which will give you a higher performance. If you don't have exactly the pieces shown on the right, don't be afraid to improvise or alter the design.

The arm has only two independent axes of movement, so that it can be built using just two motors. The motor controlling the vertical movement of the arm also operates the gripper - the arm only starts to move when the gripper has fully closed, and conversely the gripper only opens when the arm has reached the bottom of its travel.

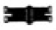
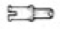
WHAT YOU NEED

2	4	6	8	12	16	
4	5	7	7	3	3	Beams

1×2	1×3	1×4	1×6	1×8	2×4	2×6	2×8	
5	2	3	8	2	5	1	3	Plates

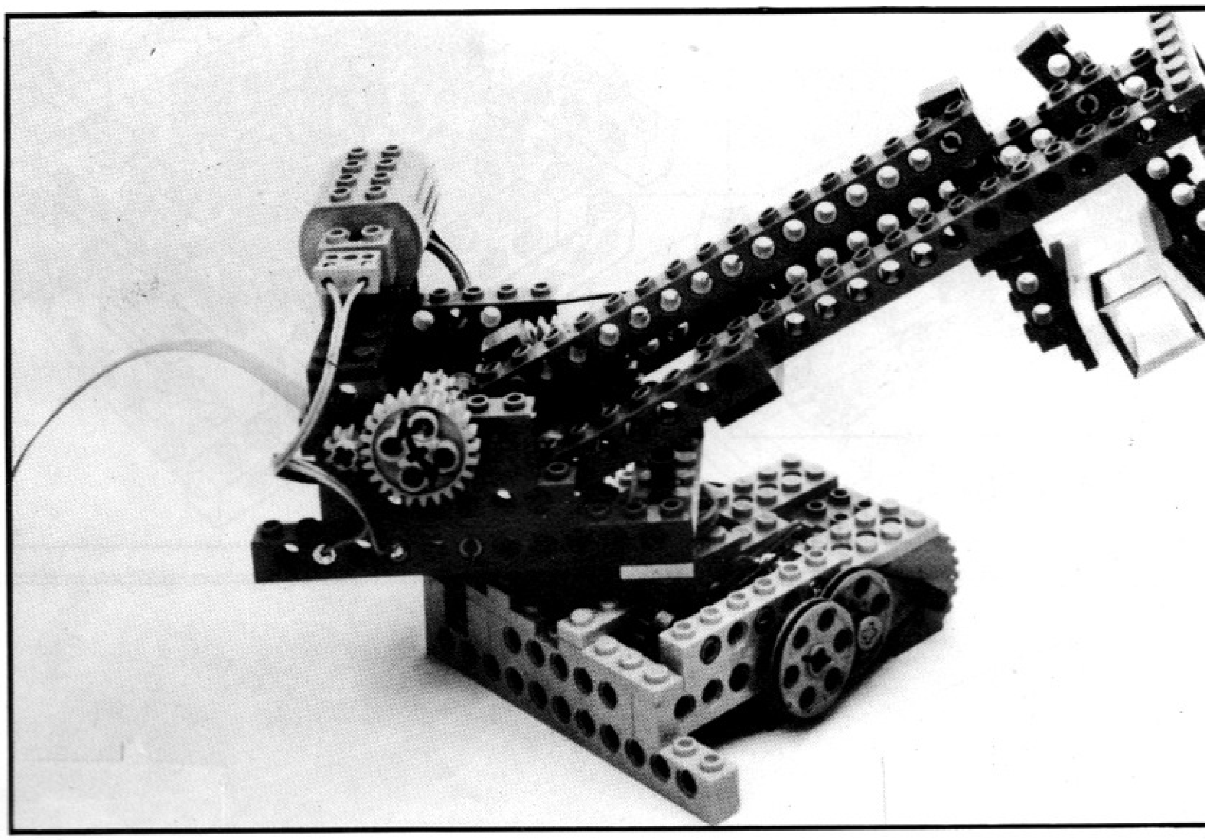
8	24	24C	40	SP	LP	
6	4	1	1	3	1	Gears

4	6	8	10	
3	1	2	2	Axles

2	5	5	18	2
				

2 motors, 3 small drive belts, 3 large drive belts, 50cm length of 8-way ribbon cable.
For the switches – belts, cooking foil, paper clips.

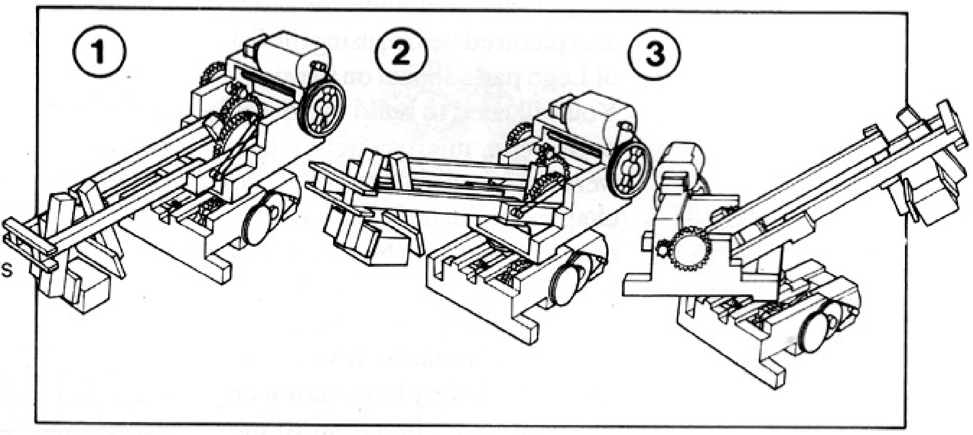
Now follow the step-by-step instructions overleaf to build the arm.



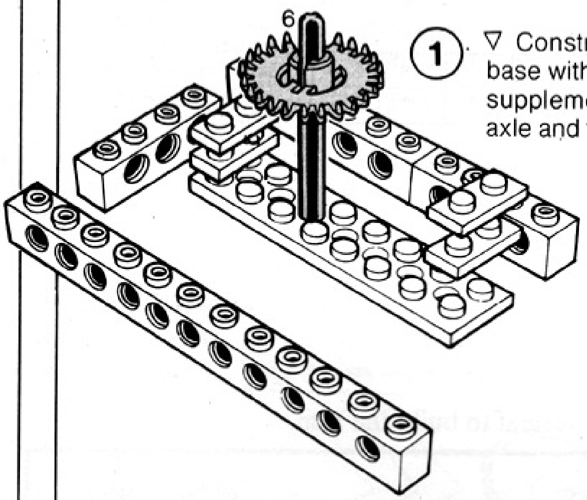
Up, round, down; up, round, down

As the arm turns at the 'waist' it can stop at any one of 36 set positions, and as it moves up and down at the 'shoulder' there are 7 or 8 stop positions from bottom to top.

The gripper closes its fingers on an object. When the fingers cannot close any more, the arm raises up to the chosen position. Then the arm turns, lowers and, as the object touches the tabletop, the grippers open. This is a 'pick-and-place' action.



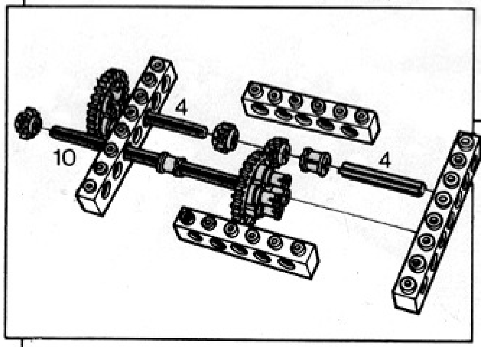
THE BASE

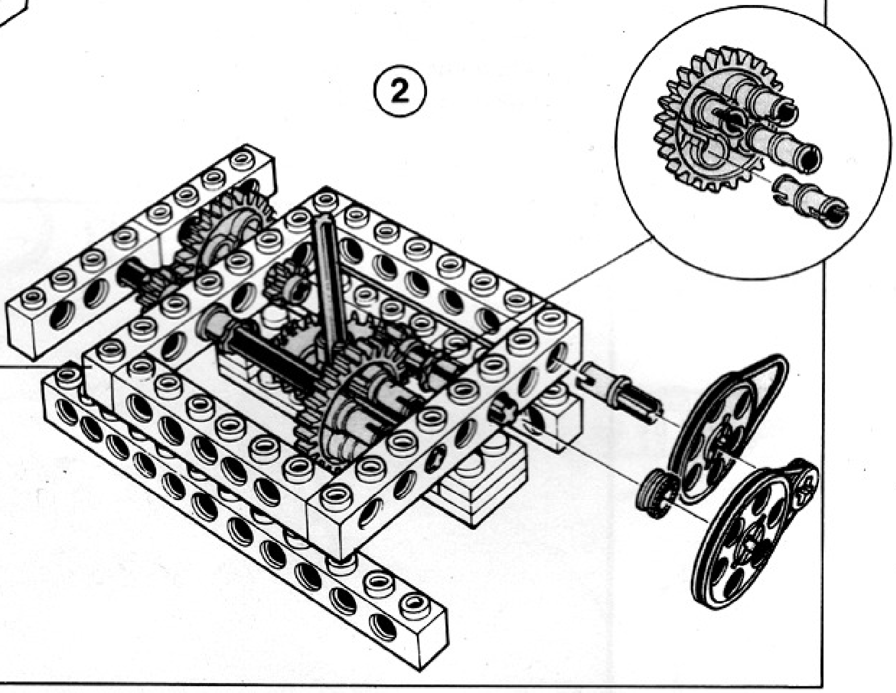


① ▽ Construct the bottom part of the base with beams and plates from the supplementary set. Add the upright axle and the gear wheel.

▽ Assemble and add the next level, with gear wheels, axles and so on, making sure you have used the correct pieces (see insets for details). Place small drive belts on the pulleys. Make sure the gears mesh properly and move freely.

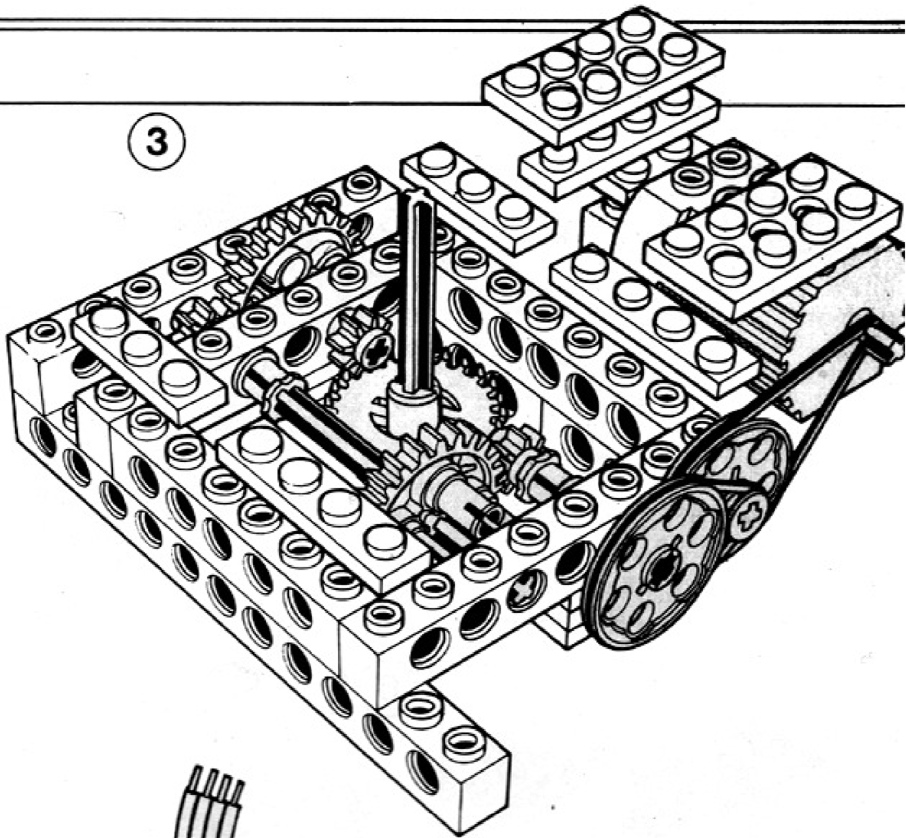
②





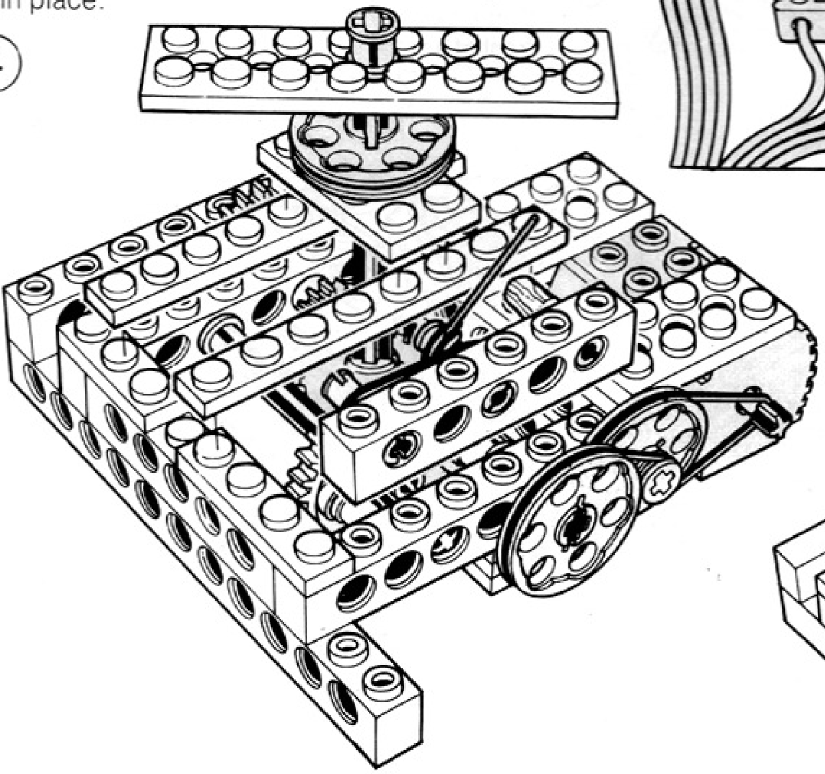
▷ Another layer of plates is added front and back. The motor unit that turns the arm is fixed at the back of the base with more plates. Hook the loose drive belt onto the motor spindle.

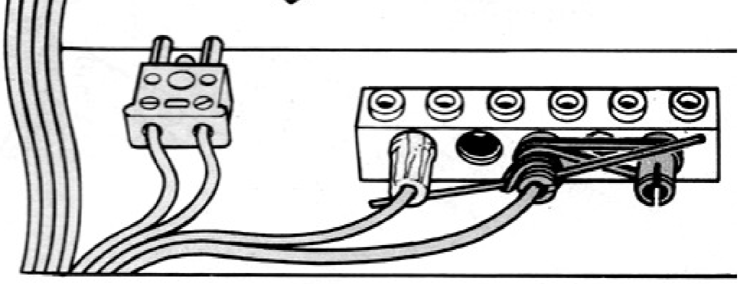
③



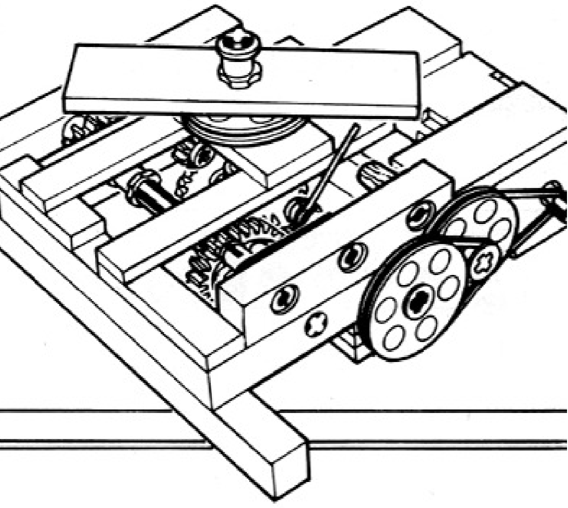
▽ As a support for the shoulder of the arm, add two 1 × 8 plates and, fitting on the axle, a 2 × 4 plate and bush. Thread the wires for the base switch sensor under the motor fixing plate and up, and fix the switch beam in place.

④





▽ The complete base unit. Put the plug into the motor and push the foil plugs into the switch pins as shown above.

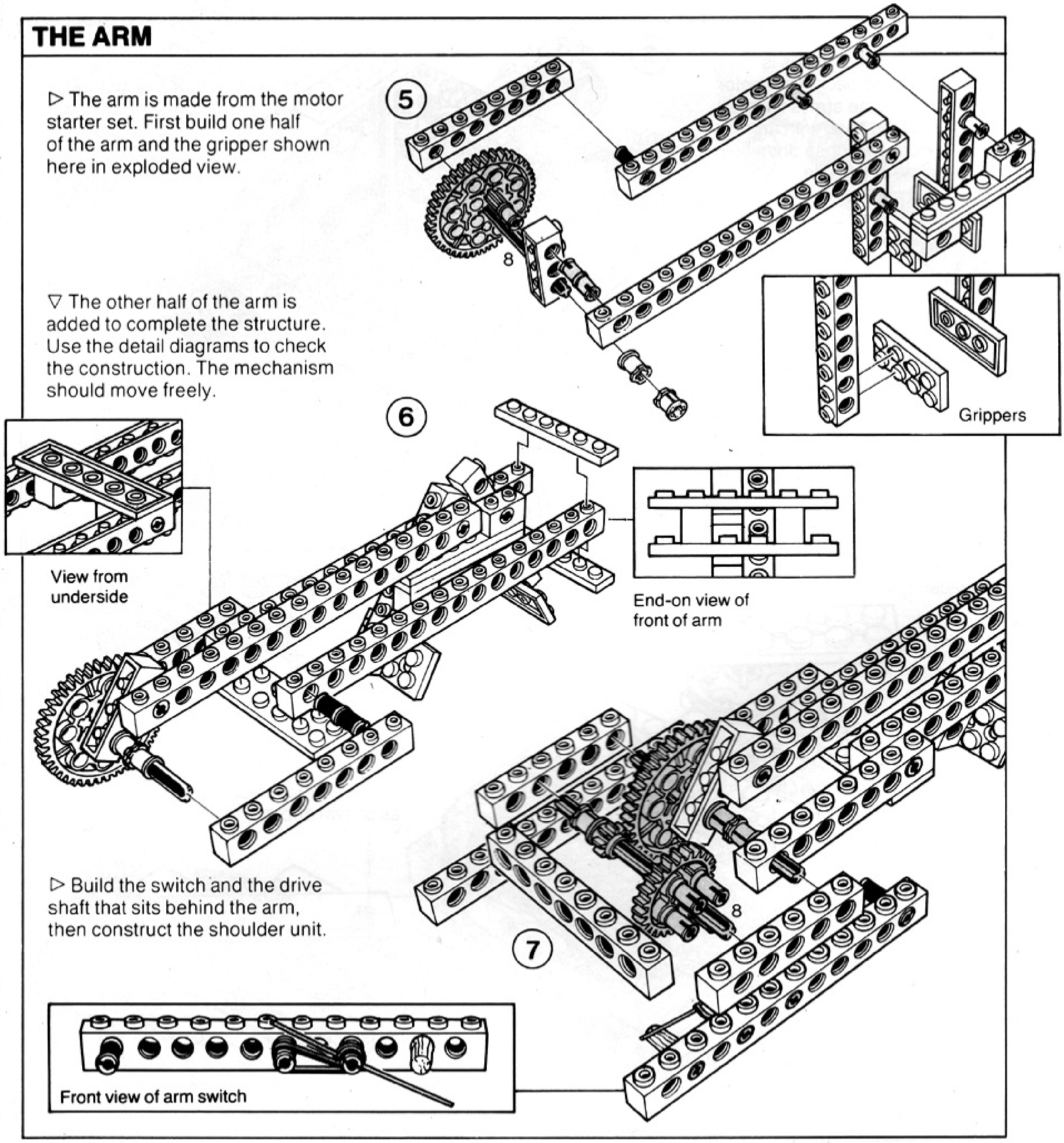


THE ARM

▷ The arm is made from the motor starter set. First build one half of the arm and the gripper shown here in exploded view.

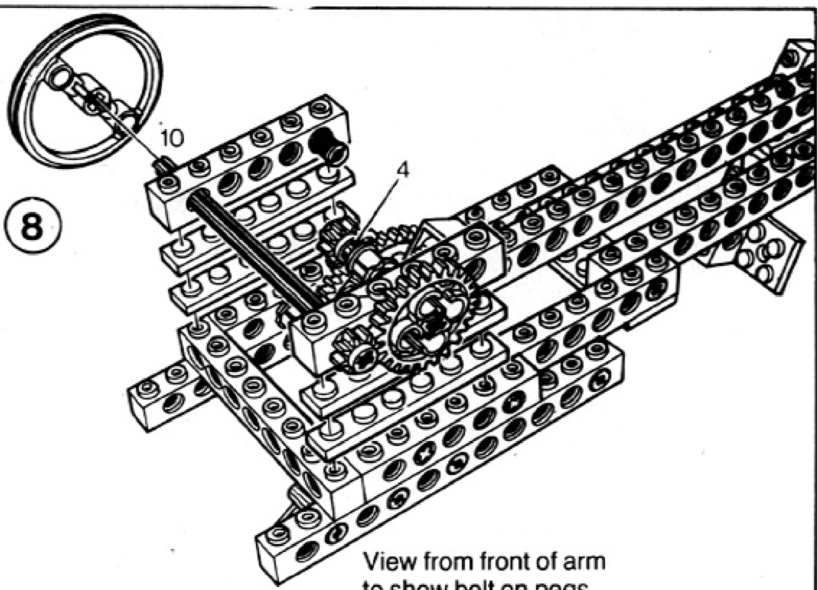
▽ The other half of the arm is added to complete the structure. Use the detail diagrams to check the construction. The mechanism should move freely.

▷ Build the switch and the drive shaft that sits behind the arm, then construct the shoulder unit.


5
8
Grippers
6
View from underside
End-on view of front of arm
7
8
Front view of arm switch

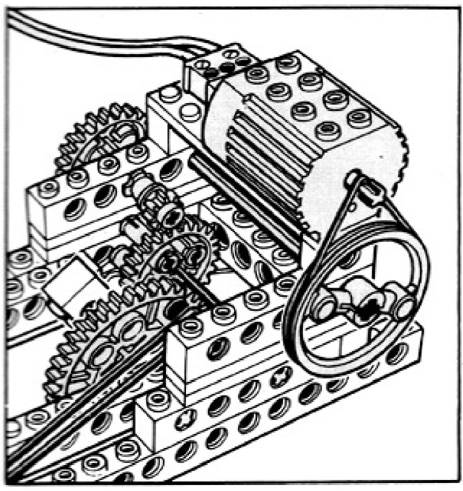
▷ Add the last layer of gears and beams to the top of the model. The gears should move freely and smoothly.

⑧

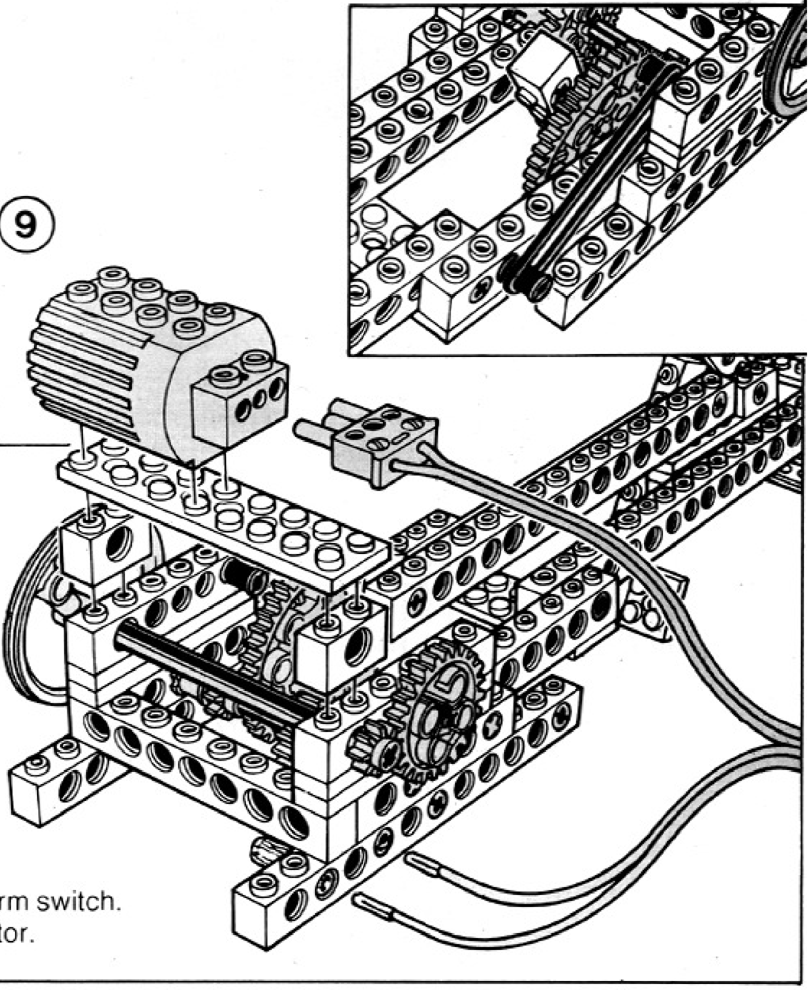


View from front of arm
to show belt on pegs

View of motor-
pulley assembly

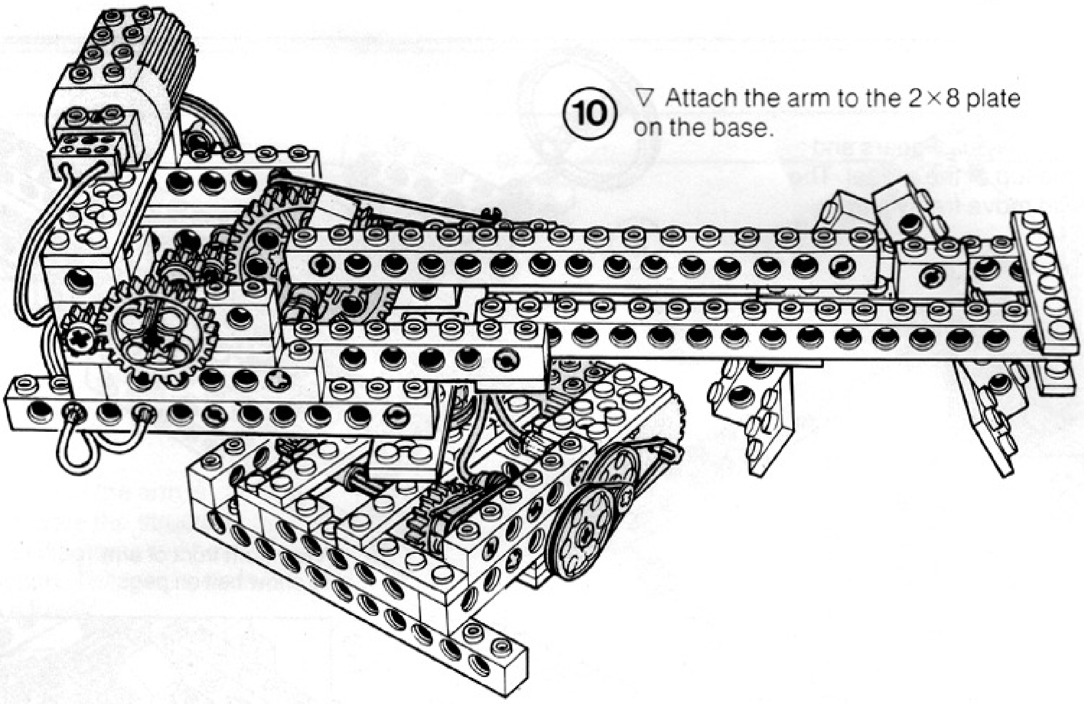


⑨



△ The shoulder motor fits onto a 2×8 plate set on two small beams on the top of the model. Use one large drive belt on the pulley and one on the pegs to help the arm move up and down.

Plug the wires into the arm switch.
Put the plug into the motor.



⑩ ▽ Attach the arm to the 2×8 plate on the base.

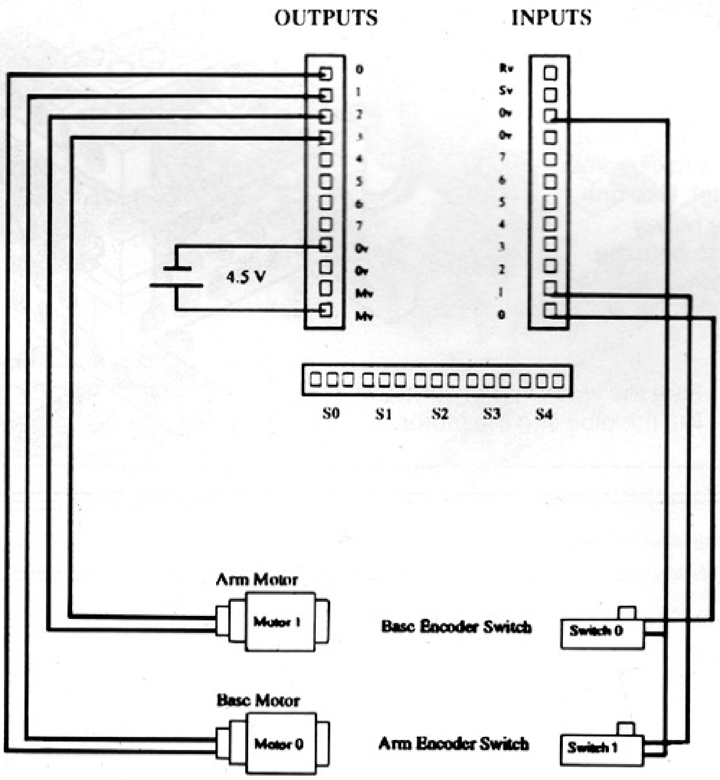
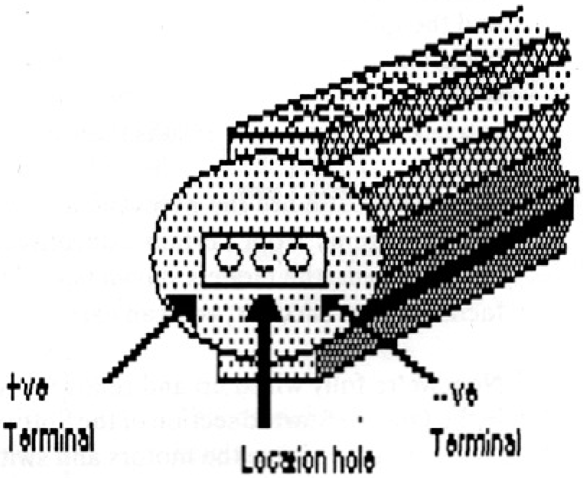

OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
Mv
Mv
4.5 V
Rv
Sv
0v
0v
7
6
5
4
3
2
1
0
S0
S1
S2
S3
S4
Arm Motor
Motor 1
Base Encoder Switch
Switch 0
Base Motor
Motor 0
Arm Encoder Switch
Switch 1

Figure 2.4 Wiring Diagram for the Mini-arm

Now that you've built the Mini-Arm model it can be connected up to the Atari Robokit Interface in order to give full control of its motors and to read the pulses from the paper clip switches. If you look at the Mini-Arm wiring diagram you'll see two labelled motors, two switches and a 4.5v battery pack. We'll start by looking at the battery connects and then move onto the motors and switches.

Since the only thing being powered in this project are the two motors it follows that the battery connection to the interface should be through the terminals labelled 'Vmtr' and '0v' on Connector 3. The 'Vmtr' terminal is the positive motor power input to the interface and the terminal labelled 0v (for zero volts) is the ground or negative terminal.

Figure 2.5 The Lego battery box



It is very important to ensure that the battery is connected the right way round so take a few moments to check your battery pack carefully for the '+' (positive) and '-' (negative) wires and connect these up to the interface as shown in the diagram - it doesn't matter which of the two adjacent '0v' terminals you use because they are effectively the same terminal - the same goes for the two Vmtr terminals.

If you are using one of the long grey tubular Lego battery packs then you'll probably find that the plug at the end isn't labelled positive and negative. Not to worry, if you've inserted the batteries in the holder as shown inside, then the positive side of the plug is on your left as you look end onto the pack with the nobbles on the top of the pack facing upwards (see diagram). It's probably a good idea if you mark this side of the plug with a red felt pen, it'll save time and confusion at a later date and will help to protect your interface.

If you're using a Lego lead, or any other joined pair of wires, carefully trace one of the wires from one end to the other and again mark each end with a red pen, this will be the positive wire, to be connected to the positive side of the plug. Now you can connect up the battery pack.

That was the difficult part! Now connection of the motors and switches is very straight forward. For the motors all you need to do is take two wires from each motor and connect them up to a consecutive pair of output terminals. In the diagram you can see we've used output terminals zero and one on connector block one for motor zero, and terminals two and three for motor one. If we stick with these connections then the slot initialisation messages explained later can be used directly, making life a little bit easier. To connect up the switches you just need to take a pair of wires from each switch. Connect one of the pair from switch zero to input terminal zero on connector block two and take the other wire to the zero volts terminal on the same connector block. For switch one, connect one wire to input terminal one, and the other to one of the zero volts terminals on the same block.

Just as an aside on the switches, if you have any Lego micro- switches (which form part of the schools range) then with a bit of ingenuity you can change the model to use them as switches instead of the paper clip ones described in the text. You can use the same arrangement of a cog with four axle shafts in it, but instead of the paper clip assembly, position the micro-switch next to the axles so that when it rotates they push on the switch causing it to open and close. If it's not possible to get the micro-switch close enough to the axles then try extending the switch by using one of the grey pegs with one half axle and the other half peg. Push the axle side into the micro-switch and slide a two by one beam onto the protruding peg. This gives a nice flat surface for the axles to push against while extending the reach of the micro-switch to give a bit more flexibility. The micros witches will give better results than the home-made paper- clip switches.

Even better still are the new optical encoders available in the Lego control sets, or as separate items from certain educational suppliers. These can be fitted even more elegantly onto the models shown here. Depending on which gear in the gearchain is attached to the encoder, you can vary the accuracy ('resolution') of movement possible.

Now we're fully wired up and ready for action! If you start up ROBOKIT as explained in the Getting Started section of the Software Guide then we can send all the appropriate messages to initialise the motors and switches and then start controlling our robot arm.

2.2. Lift Operator

This project serves as an excellent introduction to the need for feedback in control applications. If you were to attempt to move the lift between floors simply by timing the motion, the positioning would be very inaccurate and would deteriorate with every move. Instead we make use of a microswitch to count physical markers on each 'floor'. If you have the Lego shaft encoders, you can use one of them to count black-coloured bricks, which will provide greater accuracy and reliability.

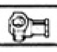
WHAT YOU NEED

2	4	6	8	12	16	
4	2	4	5	4	6	Beams

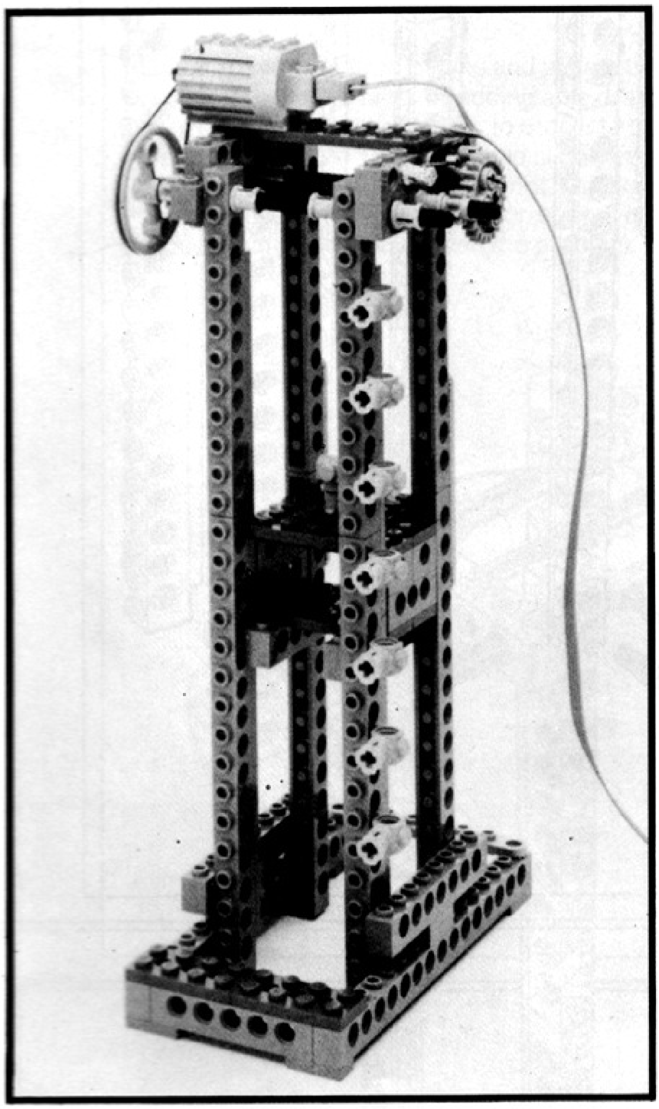
1×2	1×6	2×4	2×6	2×8	4×6	
8	8	2	4	1	1	Plates

10	12	
1	2	Axles

8	24	LP	
2	2	1	Gears

10	7	8	7
			

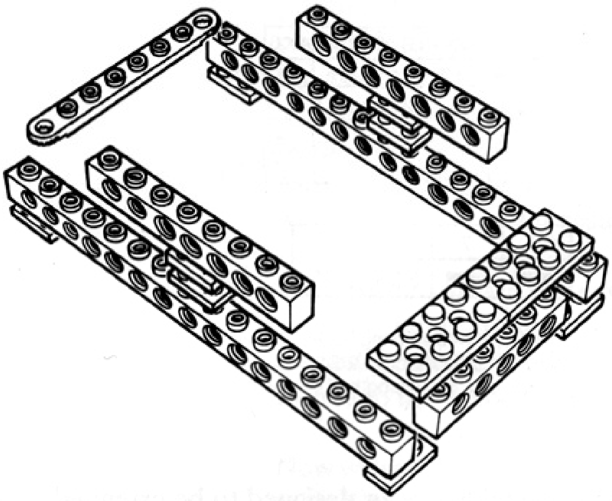
1 motor, 3 1×8 steering plates, 2 drive belts,
30cm length of string, 3 2.5cm-squares of cooking foil,
a paper clip, 50cm length of 4-way ribbon cable.



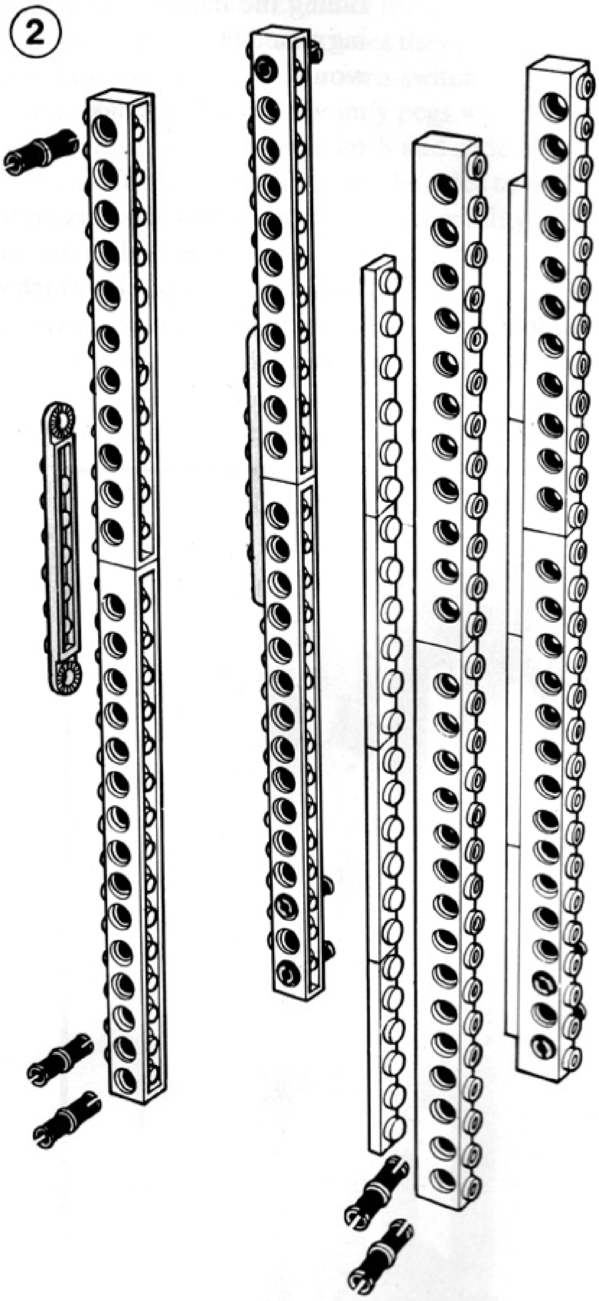
This model has been designed to be extended to any height, and could easily form part of a model garage or skyscraper.

①

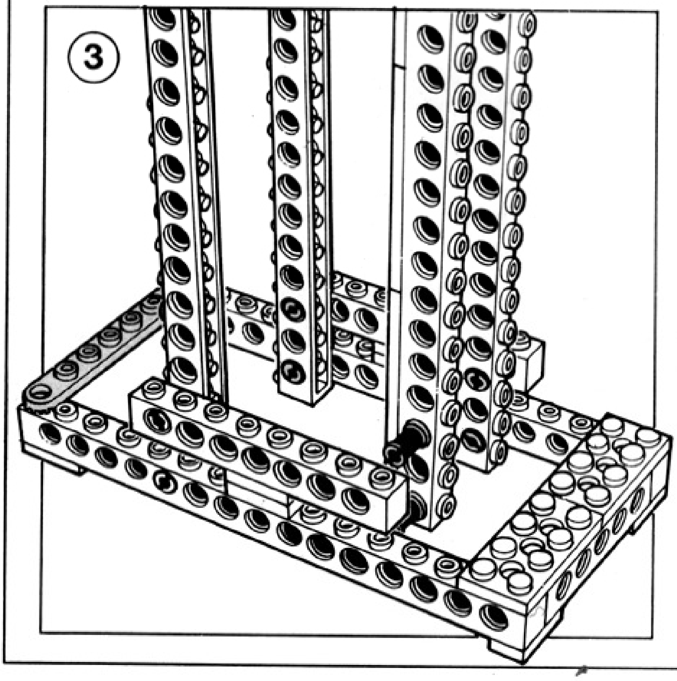
∇ The base is a simple assembly of plates and beams. A steering plate is used at the back as a cross-link.

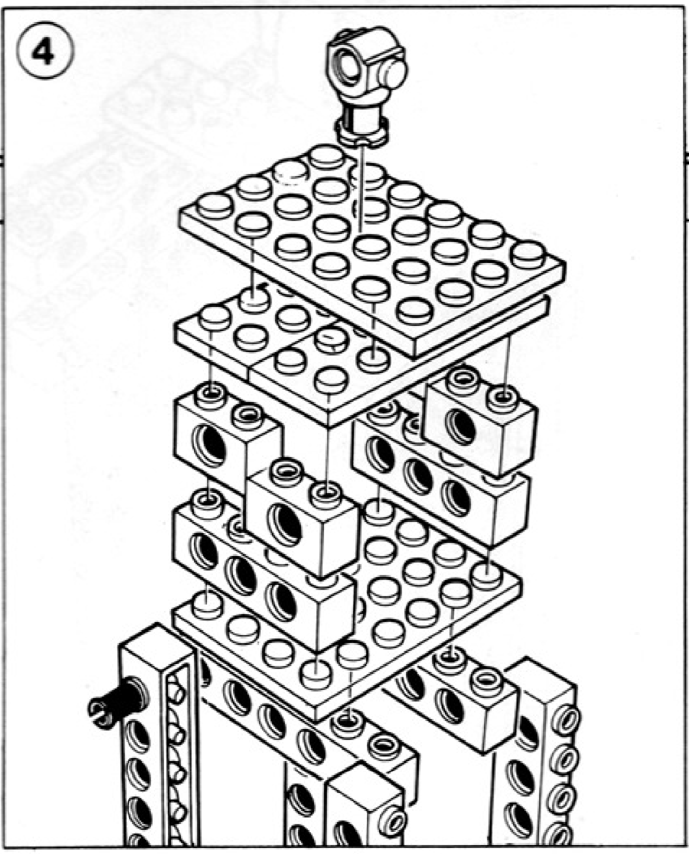


∇ Build separately the four main upright supports. Fit pegs top and bottom as shown.

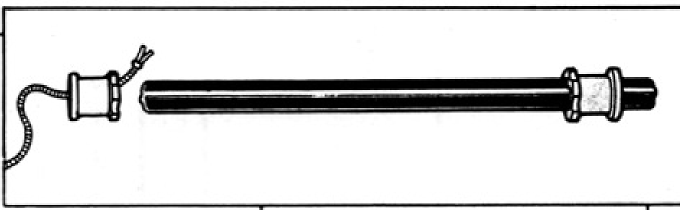


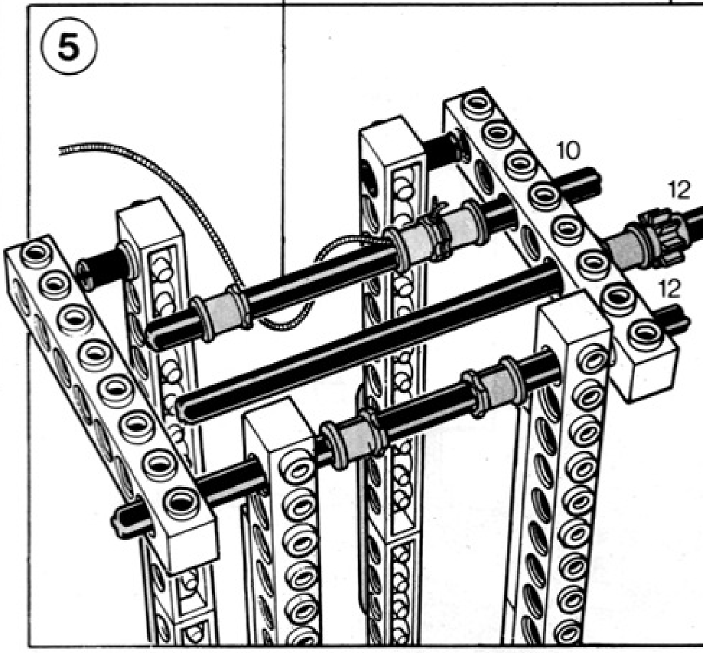
∇ Attach the supports to the base. (Only the bottom ends of supports are shown.) Push the pegs in firmly.



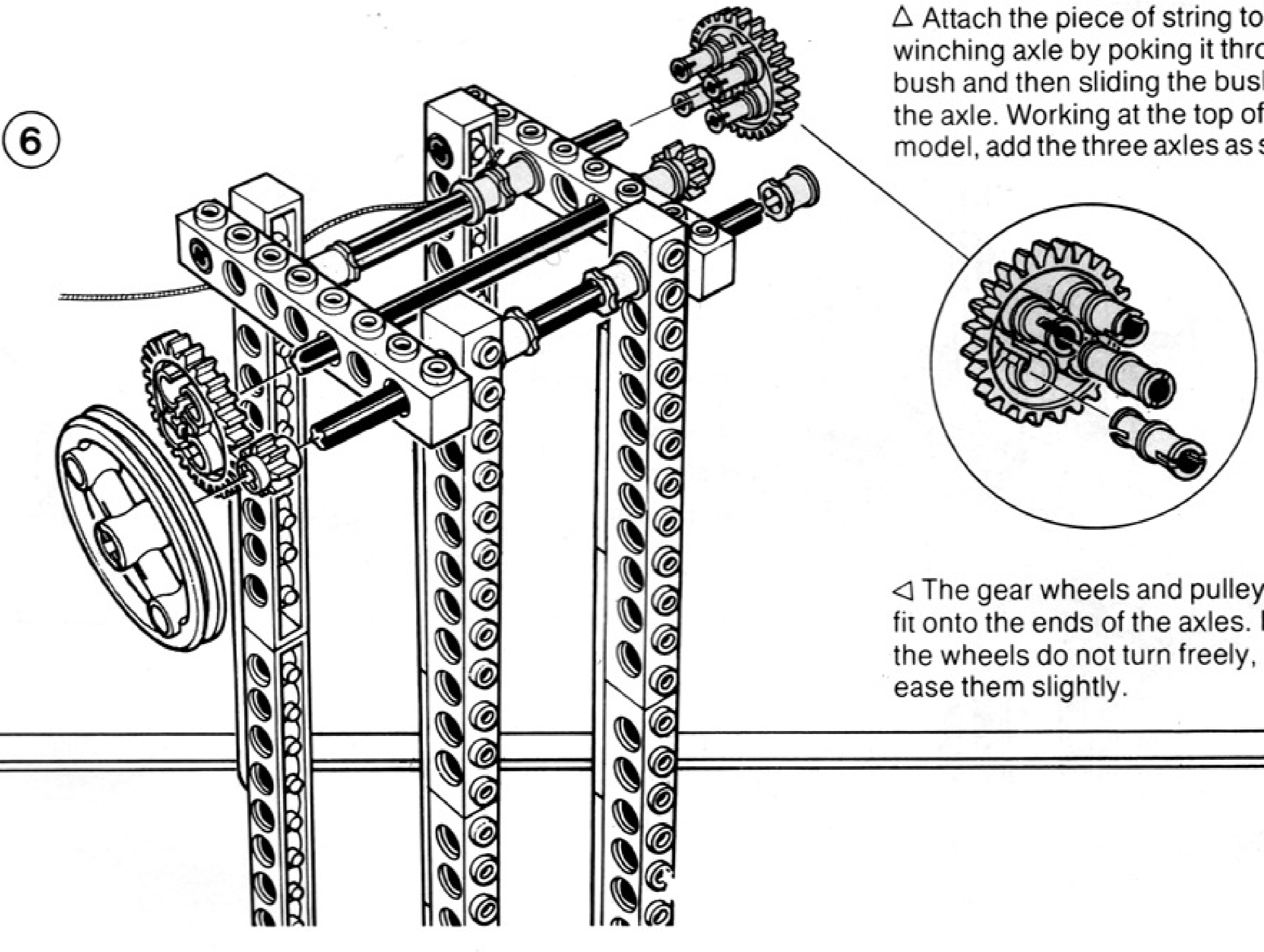


△ Build the lift car from beams and plates. If you wish, make the car taller by adding an extra layer or two of beams. Allow the car to drop into the top of the lift shaft. The car should slide freely between the supports. If it does not, check you have followed the instructions correctly. Do not proceed until you have solved the problem.



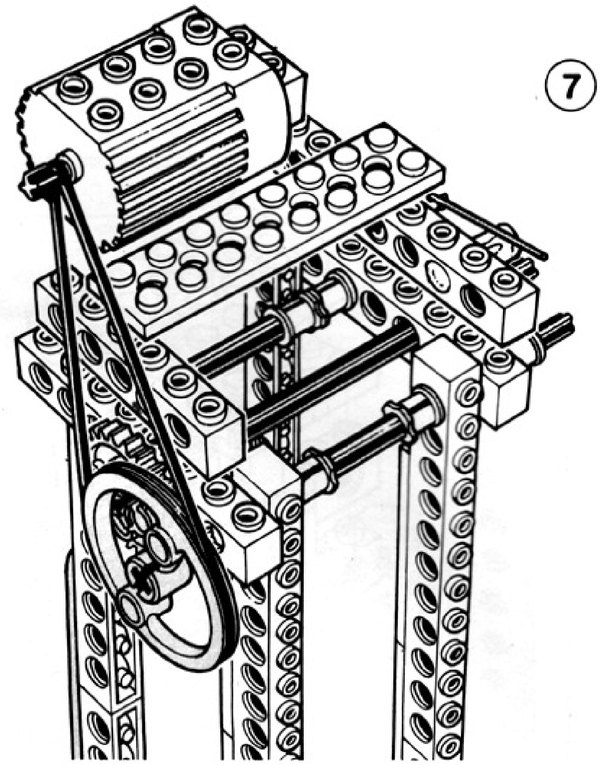


△ Attach the piece of string to the winching axle by poking it through a bush and then sliding the bush onto the axle. Working at the top of the model, add the three axles as shown.

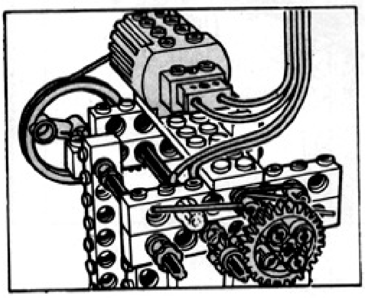


◁ The gear wheels and pulley wheel fit onto the ends of the axles. If the wheels do not turn freely, ease them slightly.

⑦

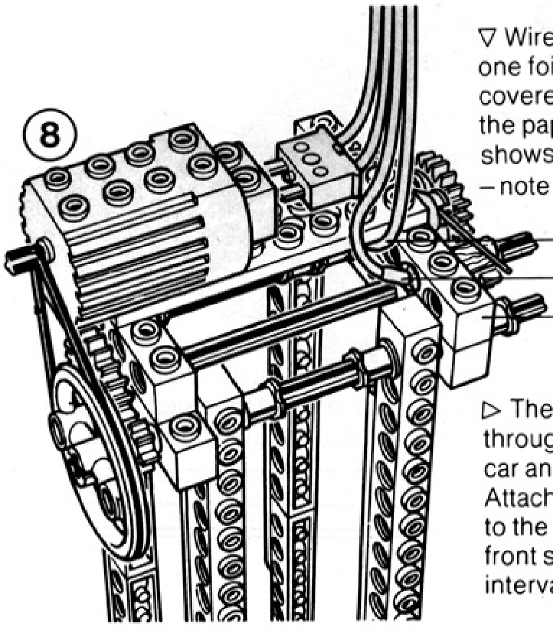


△ Add the switch beam, plain beam, cross-plate and motor. (See left for details of how to make the switch.) The paper clip on the switch should be pulled over to be above the foil. Fix the belt between motor and pulley wheel.



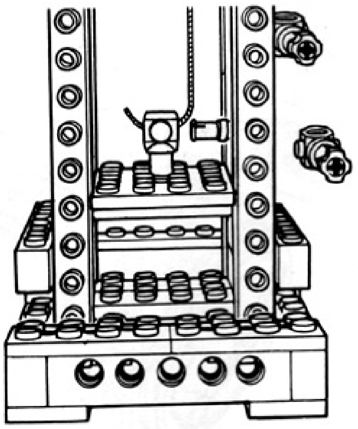
⑧

▽ Wire up the motor and switch, with one foil plug pushed into the foil-covered pin and the other plug into the paper clip pin. The inset shows the switch from the far side – note the position of the paper clip.



⑨

▷ The loose end of the string goes through the toggle on top of the car and is secured with a pin. Attach toggles (or small plates) to the outside edge of one of the front supports, at 2-hole intervals, to indicate each level.



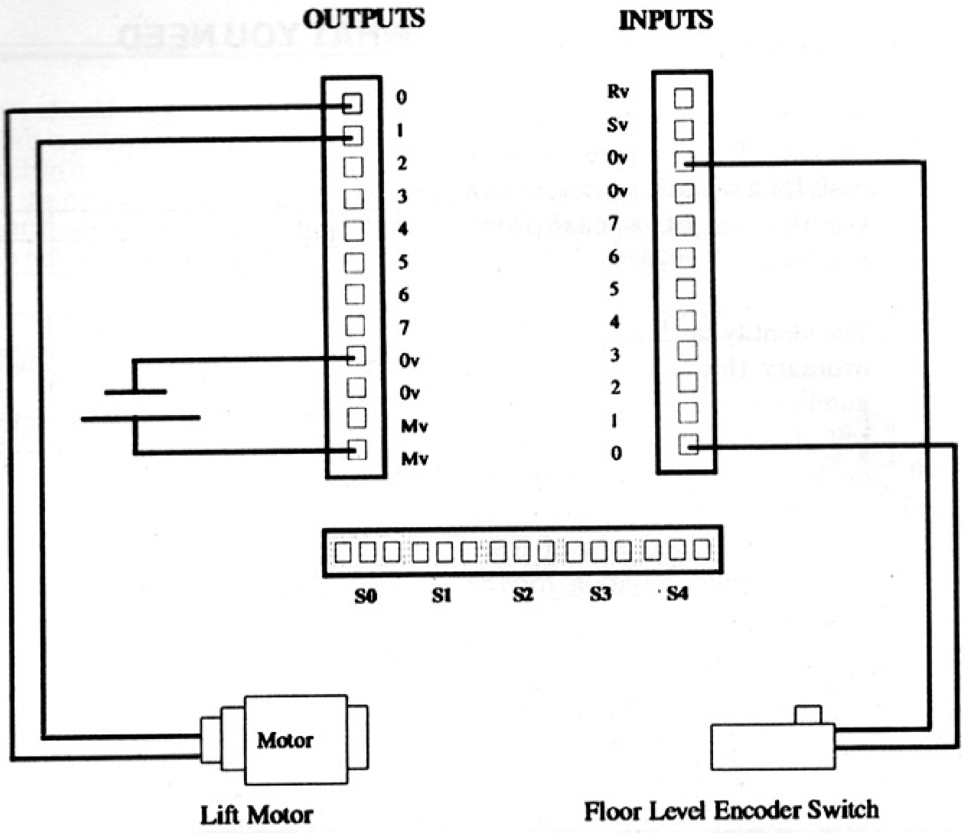

OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
Mv
Mv
Rv
Sv
0v
0v
7
6
5
4
3
2
1
0
S0
S1
S2
S3
S4
Motor
Lift Motor
Floor Level Encoder Switch

Figure 2.6 Wiring diagram for the Lift Operator

2.3. Card Reader

This simple device allows a computer to identify a cardholder by reading the number encoded on the card. It could be used as the basis for a security system, or as a simulation of a cashpoint machine.

The identity cards are made from ordinary thick cardboard. The number is encoded in binary as a row of punched holes. The motor on the card reader draws the card though the machine at a constant speed. A second row of holes provide synchronisation for the data.

WHAT YOU NEED

2	4	6	8	12	16
4	2	2	2	4	4

Beams

1×2	1×6	2×4	2×6	4×6
8	6	1	3	1

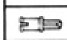
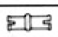
Plates

6	8
2	3

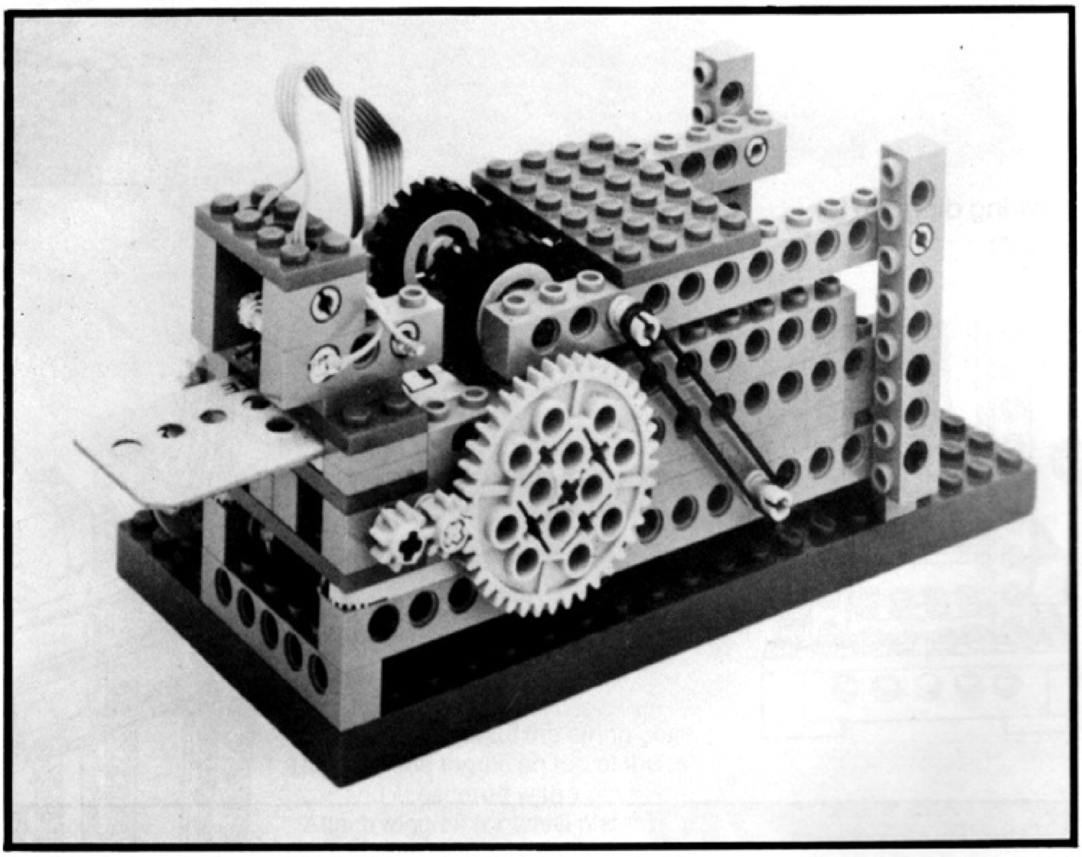
Axles

8	24	40	SP
3	1	1	1

Gears

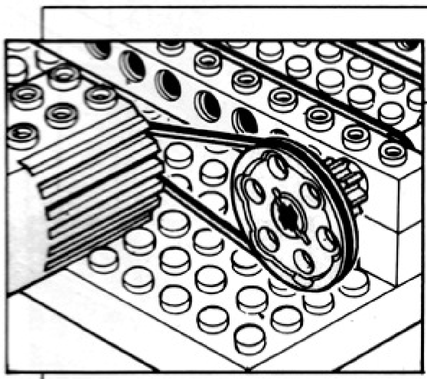
1	10	4	4
			

1 motor, 4 small wheels, 1 10×20 base plate, 6 drive belts, 1 1×8 steering plate, 2 1×2 smooth plates, 50cm length of 6-way ribbon cable, 6 2.5cm squares of cooking foil, 2 paper clips.

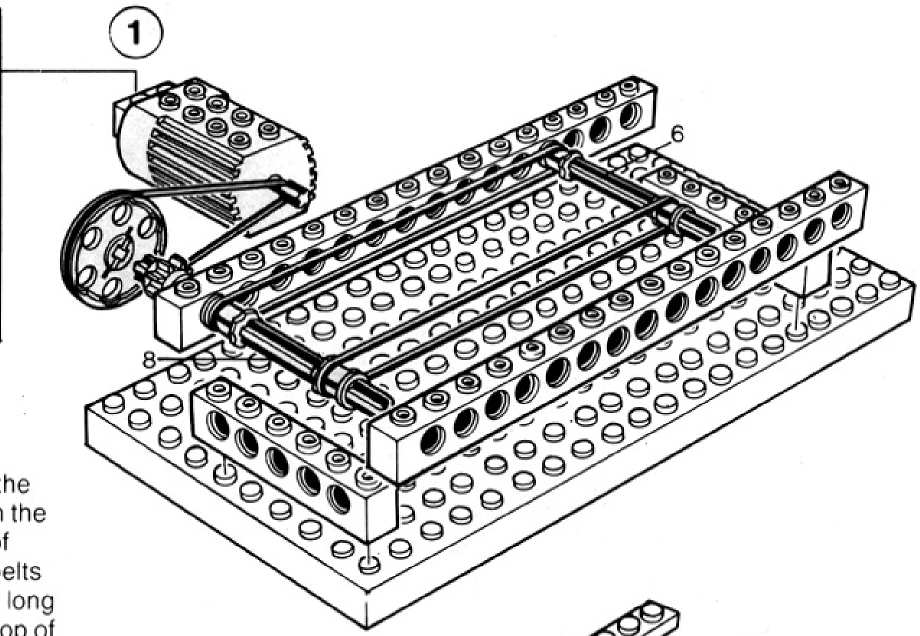


This is a security check!

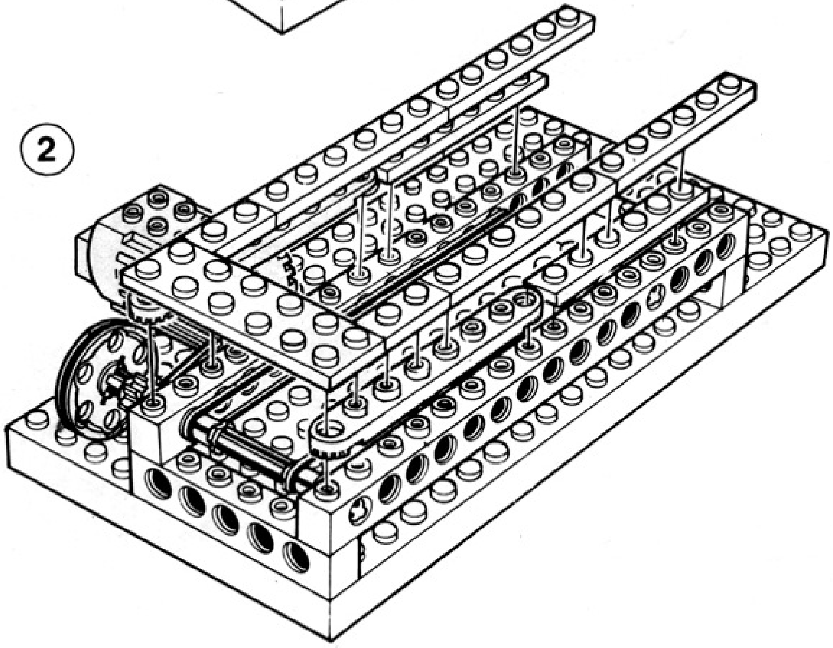
Provided the card is put in the correct way round, the right-hand switch in the reader is opened and the computer turns the motor on. The card is drawn in by the roller wheels and the holes pass under the switches. The left-hand row of holes allows the computer to sense how far the card has moved through the machine. The right-hand row of holes contains the binary encoded number. When the computer has finished reading the code, the motor is left running for a short while and the card is fed out of the reader. The motor is then turned off.



Detail of motor attachment to base plate



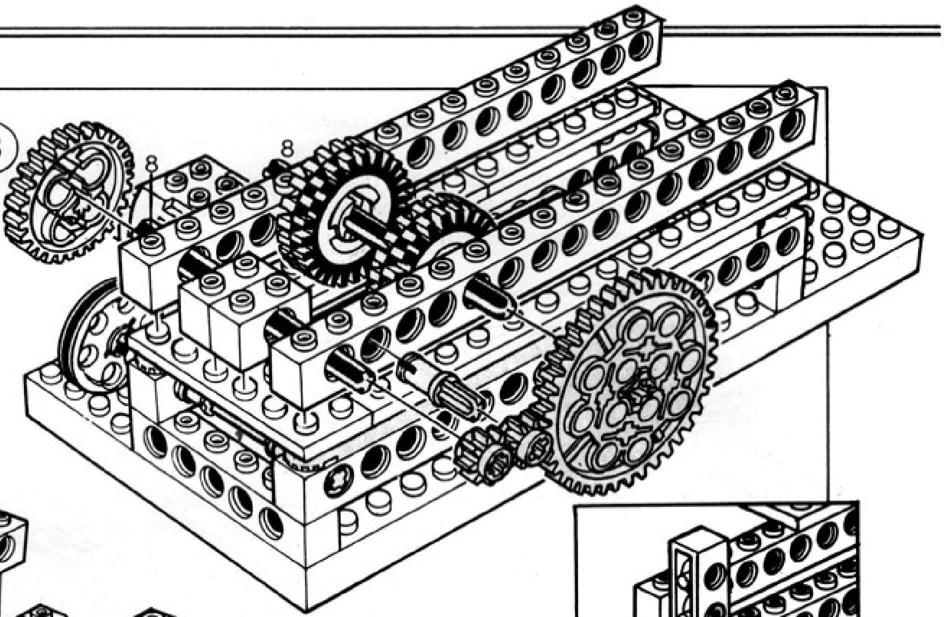
▷ Fix the two cross-beams on the base blate. Push the bushes on the axles, slide the axles into one of the long beams and place the belts over the bushes. Add the other long beam and fix the assembly on top of the cross-beams. Add the gear wheel, pulley, motor and drive belt.



▷ Add the steering plates and other plates as shown.

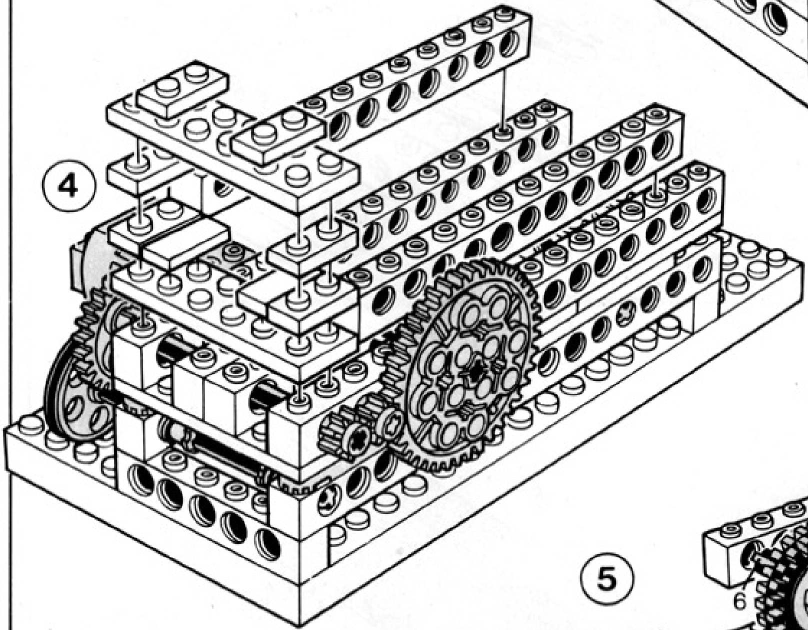
▷ Make up the next level of beams, axles, gears and small wheels, and place it in position on the first level.

③

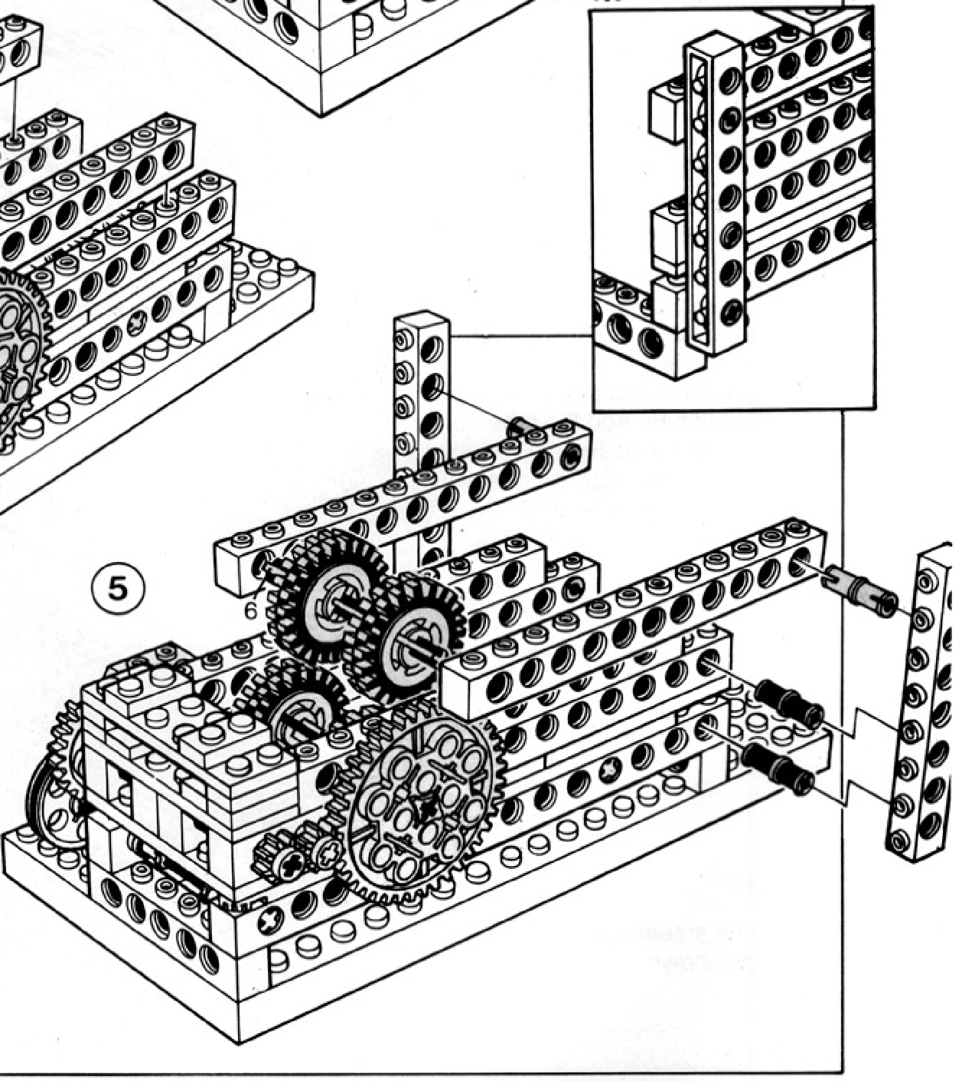


▽ Add the two 12-unit beams and the plates that at the front of the machine make up the slot for the card.

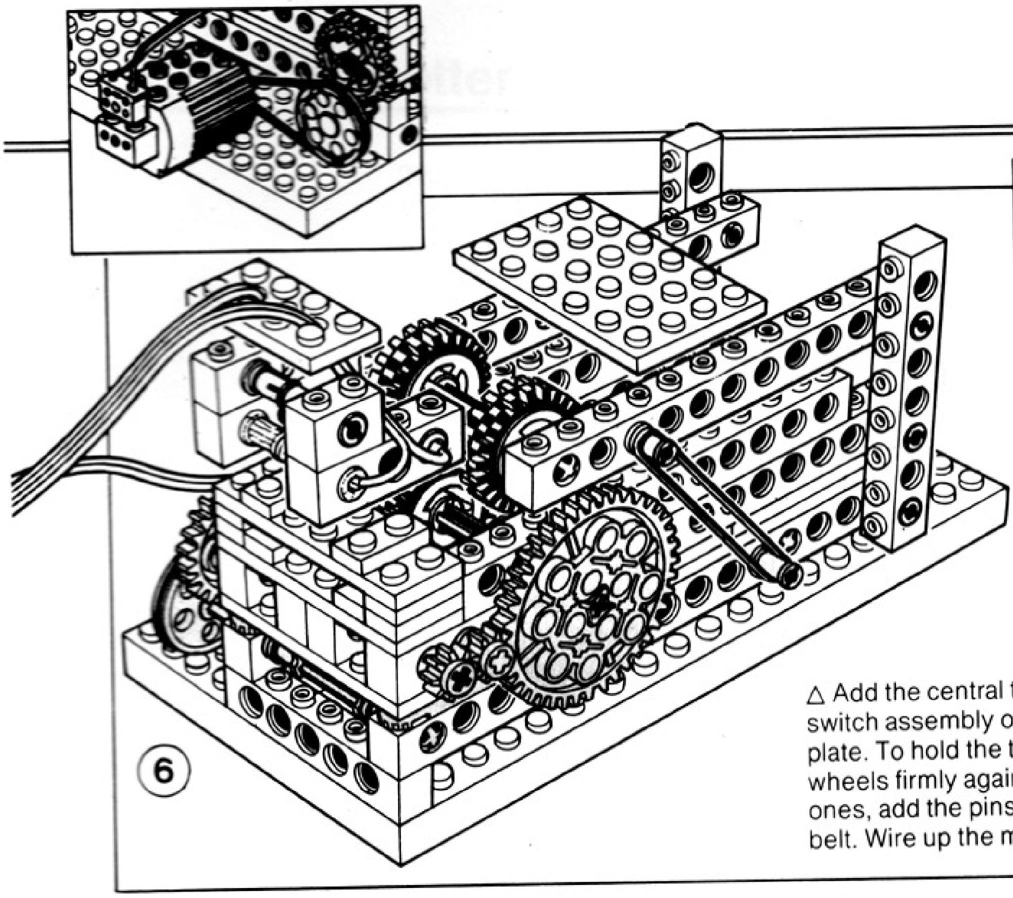
④



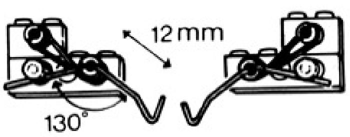
⑤



▷ The second pair of small wheels are supported in a frame which hinges on two upright beams placed at the back of the machine.



△ Add the central top plate and the switch assembly of beams and cross-plate. To hold the top card-roller wheels firmly against the bottom ones, add the pins and tension belt. Wire up the model.



△ The switches are of a different design to switches used elsewhere in the book. The arms of the paper clips should be set to the angle shown.

MAKING THE CARDS

You will need a sheet of stiff card, a hole punch, scissors, ruler and pencil.

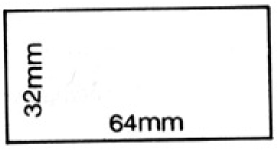
1 From the sheet of card, cut some pieces the size of a LEGO 4×8 plate (some 64×32mm).

2 Mark the back of the card as shown. Where you have made crosses, punch a hole in the card. The centre of the holes should be over the centre of the crosses. Cut out the right-hand end hole to make a slot.

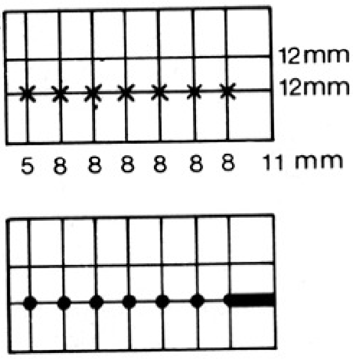
3 Number the other points on the card as indicated. To give the card its identity, punch holes at the correct points. The card's number is given by all the punched numbers added together. (In a binary number, values double from right to left, 1, 2, 4, 8 etc.)

To make the card run more easily in the reader, snip off the corners with scissors.

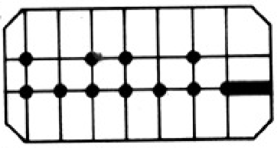
1



2



3



32 +8 +4 +1=45

Operating hints and tips

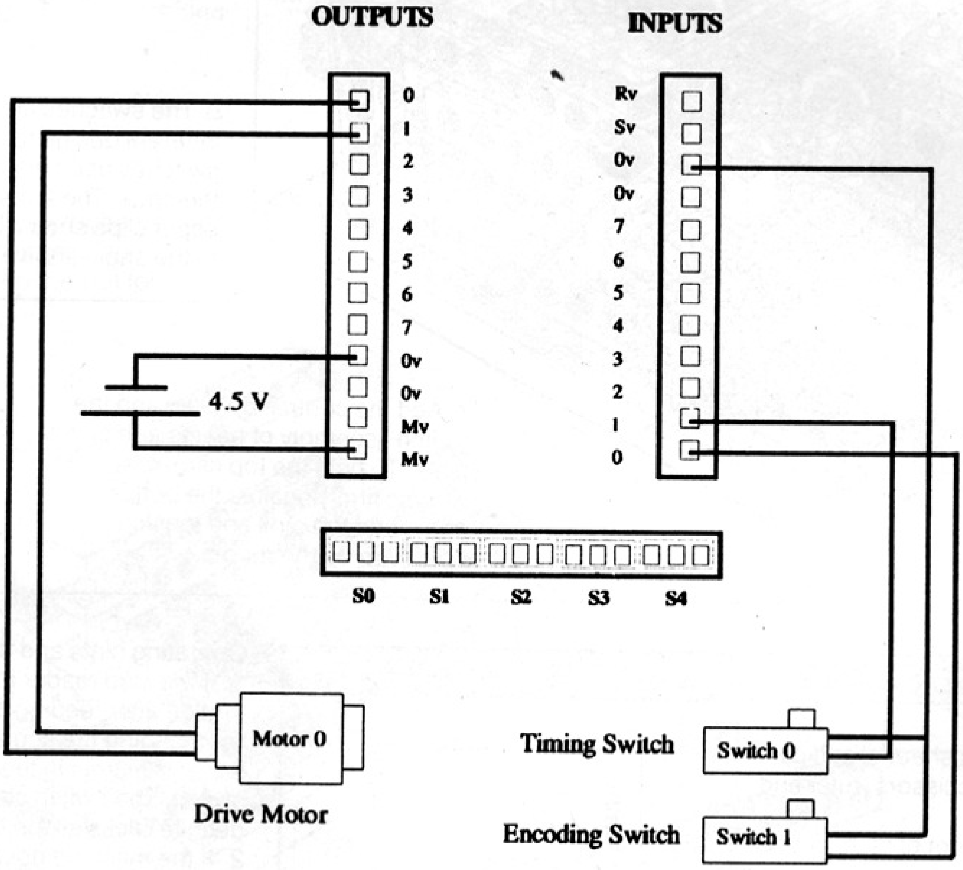
1 If the card reader motor does not turn off after feeding a card through, try changing the angle between the paper clip arms in the *left-hand* switch. The switch should make definite clicks as the card passes.

2 If the machine does not read the card correctly, try changing the angle between the paper clip arms in the *right-hand* switch.

3 If the motor does not start when you poke a card into the slot, check that only the right-hand switch is opening. If the switch is opening, check you have wired up the switches, motor, interface, and so on, correctly.

4 If your cards do not pass easily through the machine, trim them carefully with scissors or a Stanley knife so that they are a loose fit, or experiment using card of a different thickness.

Figure 2.7 Wiring diagram for the card reader


OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
Mv
Mv
Rv
Sv
0v
0v
7
6
5
4
3
2
1
0
4.5 V
S0
S1
S2
S3
S4
Motor 0
Drive Motor
Timing Switch
Switch 0
Encoding Switch
Switch 1

2.4. Drum Plotter

This is a very crude model of the kind of drum plotter used to create large scale computerised plots of maps, architects drawings and integrated circuit layouts.

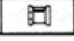
The X and Y movements are effected by rotating the drum and moving the pen laterally. Feedback is used on both motors to keep track of position. A home-made solenoid is used to lift the pen on and off the paper, which should be taped to the drum for best effect.

WHAT YOU NEED

2	4	6	8	12	16	
9	7	3	4	3	8	Beams

1×2	1×4	1×6	1×8	2×4	2×6	4×6	
5	3	7	6	8	1	1	Plates

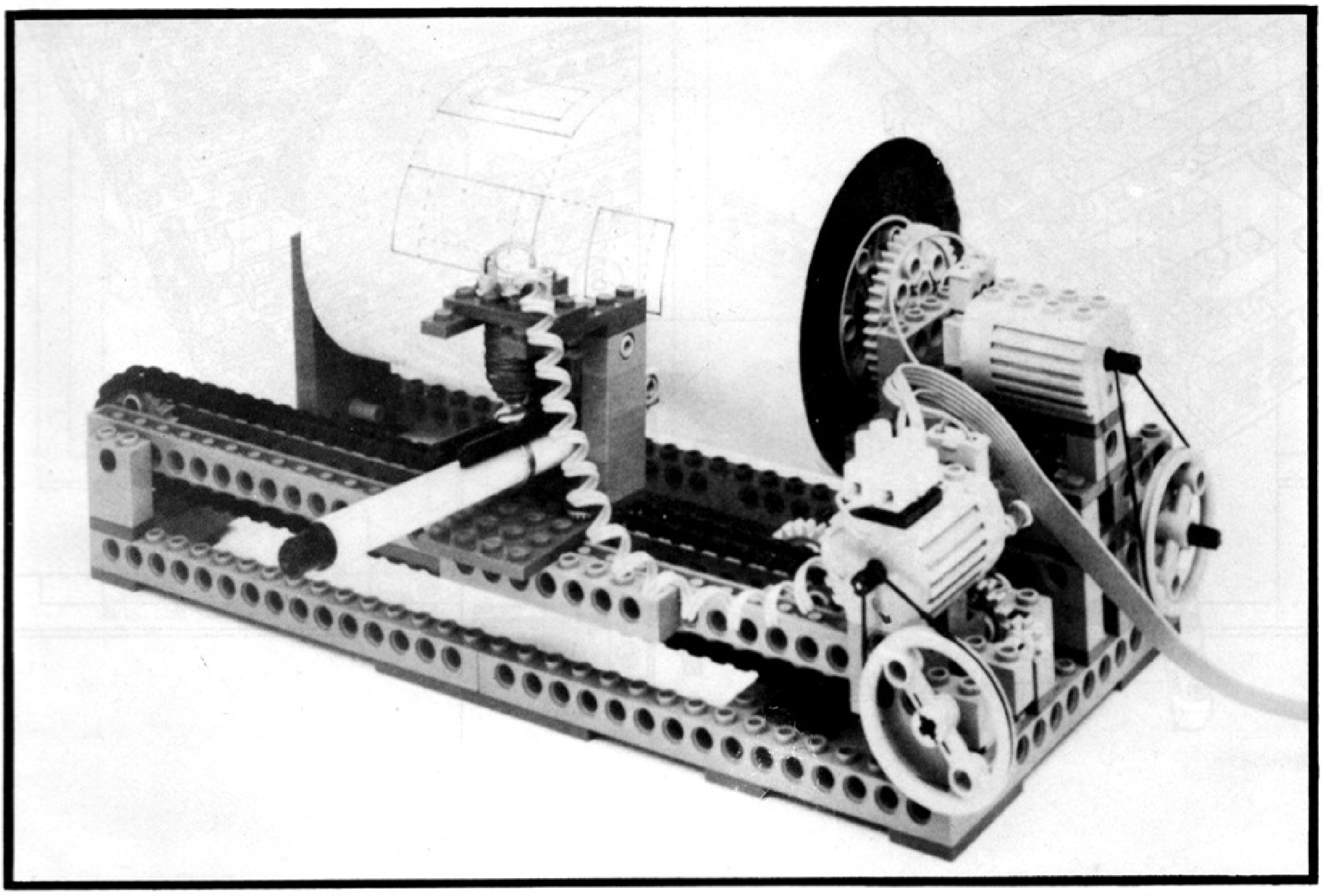
8	24	40	SP	LP	
7	6	2	4	3	Gears

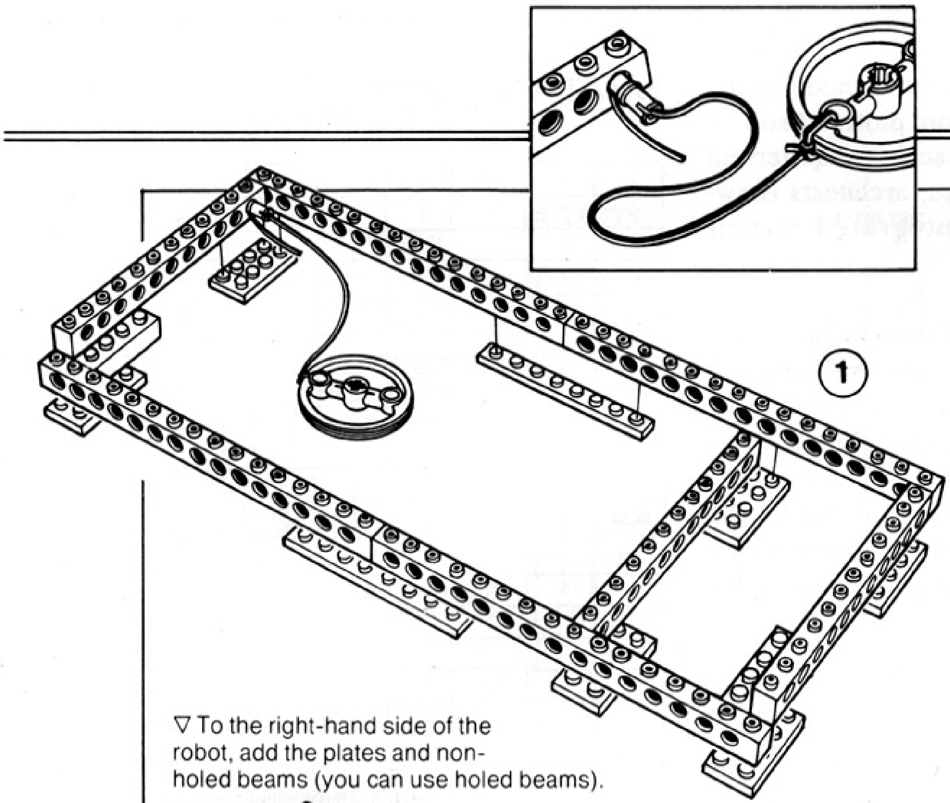
4	22	2	2	5	3	2
						

2	4	8	Non-holed
11	10	2	beams

2	4	6	8	
2	1	7	2	Axles

2 motors; 4 small belts, 3 medium; 2 large wheels; 4 plugs, 2 battery boxes; 50cm length of 10-way ribbon cable, 1 paper fastener, paper clips, 1 steel metric-5 25mm-long bolt with nut; 34 s.w.g. enamelled copper wire (obtainable from electronics shops); thin card, double-sided sticky tape, Blu-Tack, 2-way terminal block.

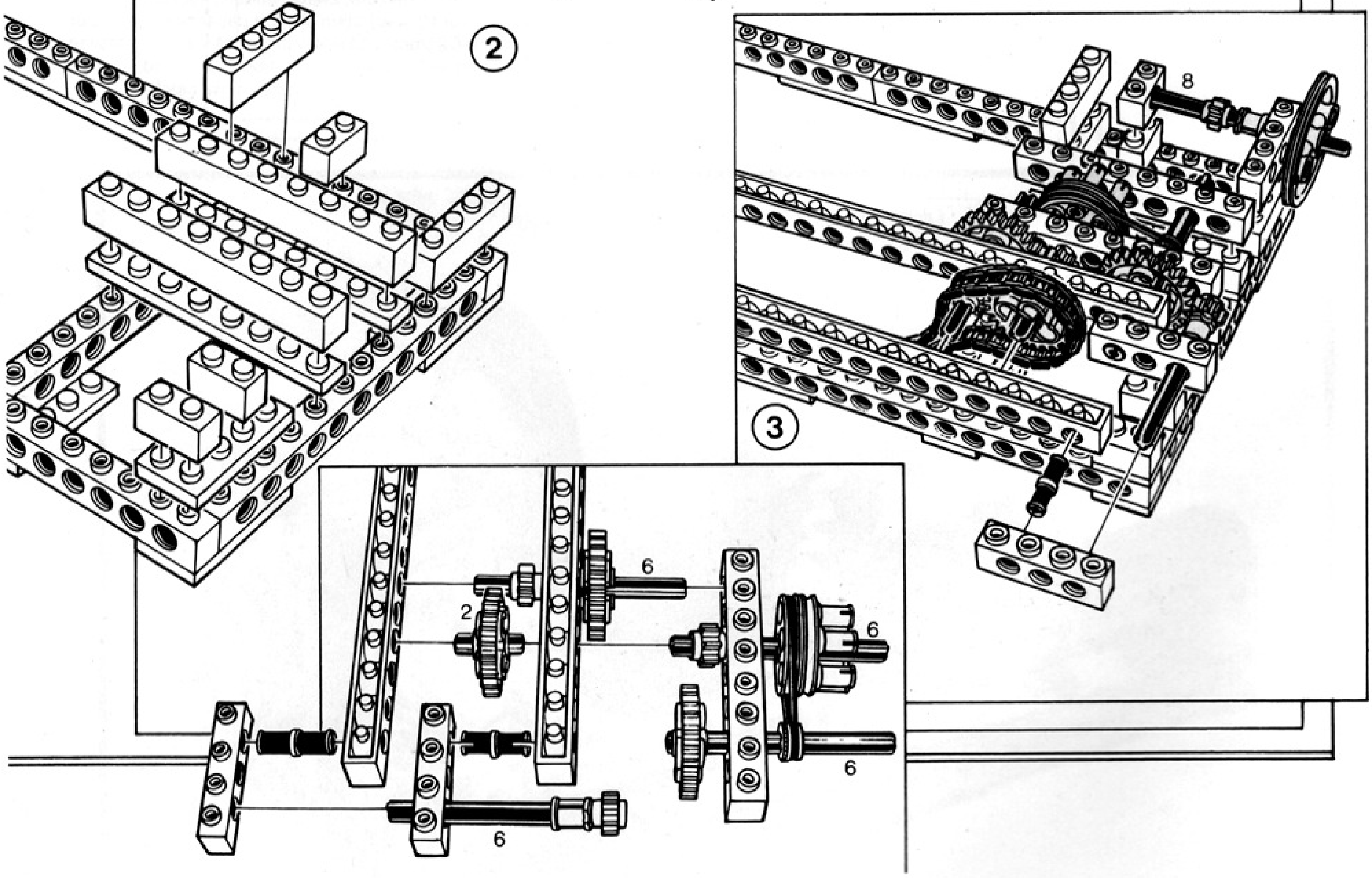


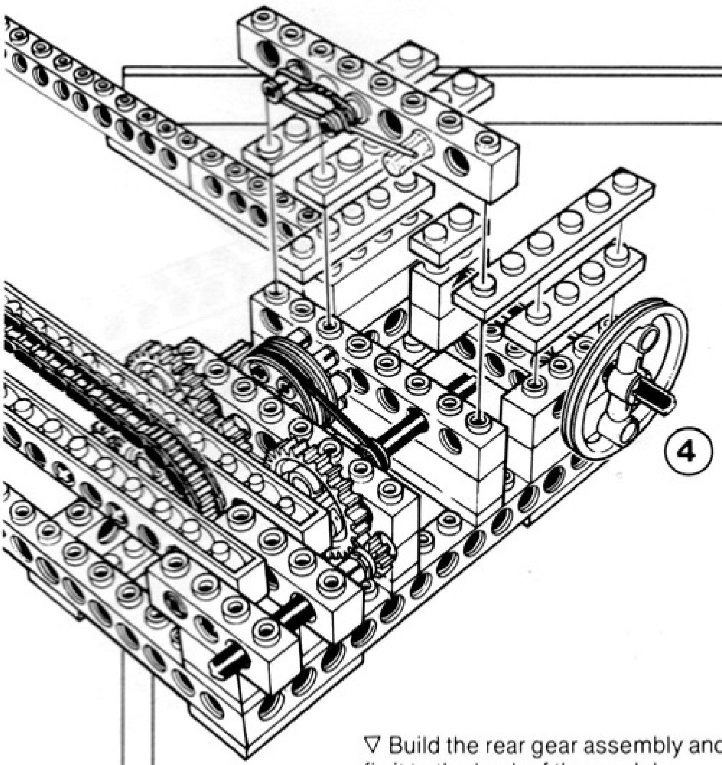


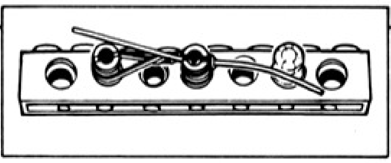
◁ Build the frame for the base. Break a medium belt, tie one end through a rim-hole on the pulley and make a knot, and secure the other end to the cross-beam using a pin.

▽ To the right-hand side of the robot, add the plates and non-holed beams (you can use holed beams).

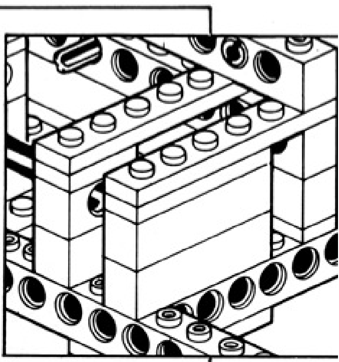
▽ Build the gear assembly as below (inset). Use a small belt on the pulley. Place the chain round the 24-tooth cog which sits between the two 16-unit beams.





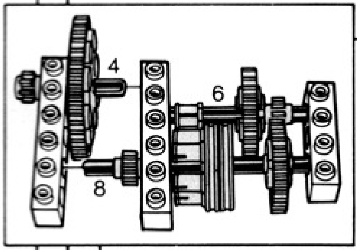


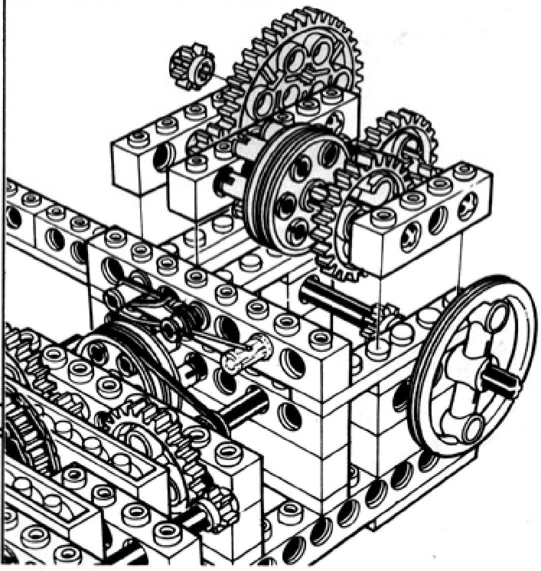
◁ At the back of the assembly, add the plates and switch for the belt drive. Remember to pull the paper clip arm over above the foil pin before securing the switch. The paper clip should be slightly curved on what will be the left-hand side of the switch (see inset).



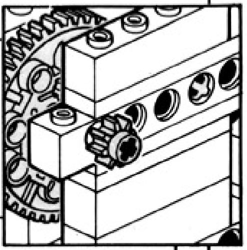
Rear of drive assembly

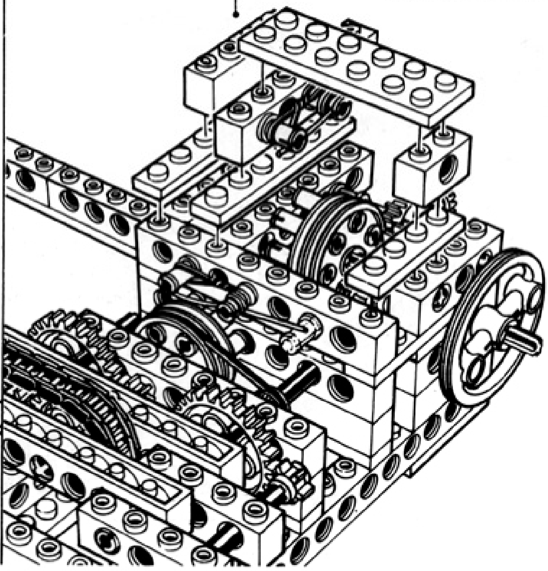
▽ Build the rear gear assembly and fix it to the back of the model. The gears must run smoothly.



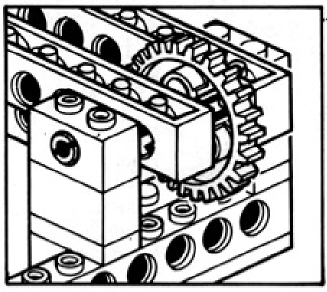


▽ The rear 4-unit-long beam is not holed. Add the drum drive switch, beam and plate as shown. As in step 4, pull the paper clip arm over the foil pin and note the left-hand part of the clip must be curved.

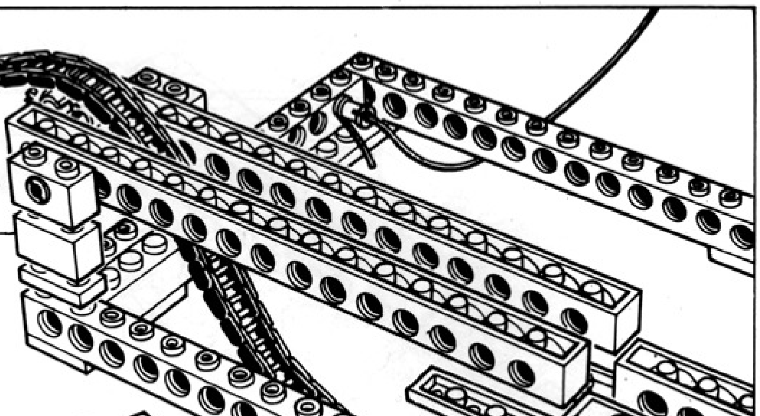




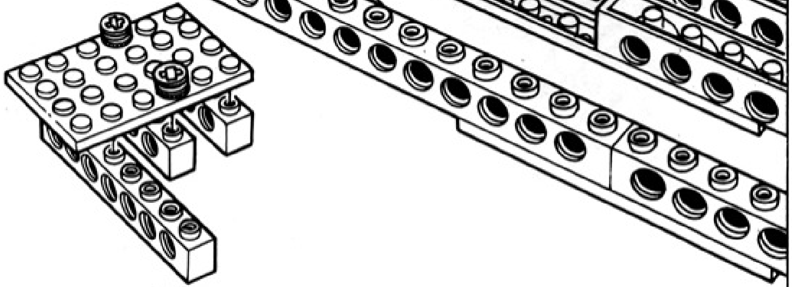
▷ Working again at the front left-hand side, add the other half of the chain drive assembly. Allow the chain to pass under the 24-tooth cog wheel.



⑦

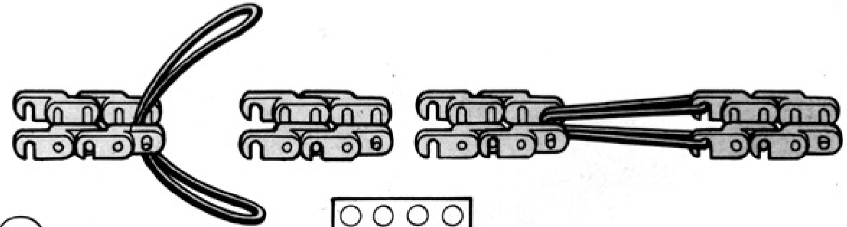


⑧

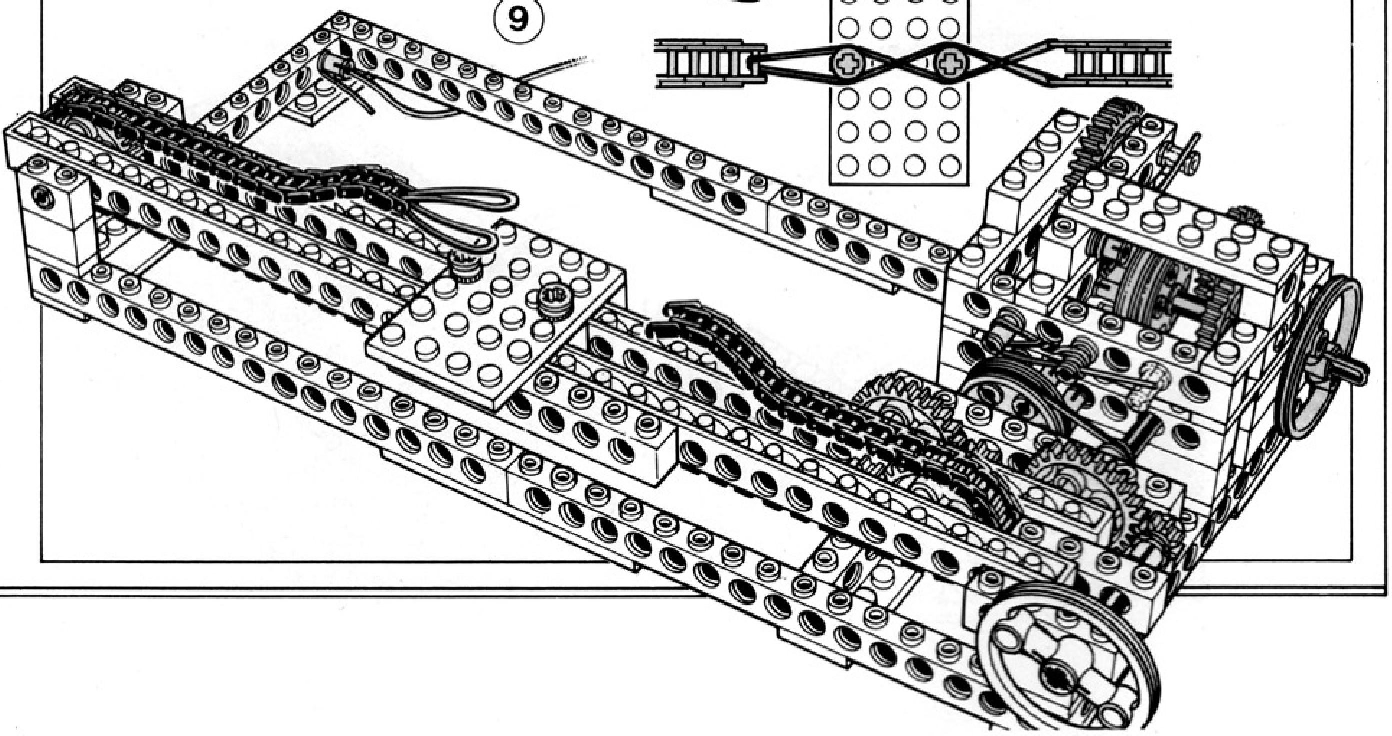


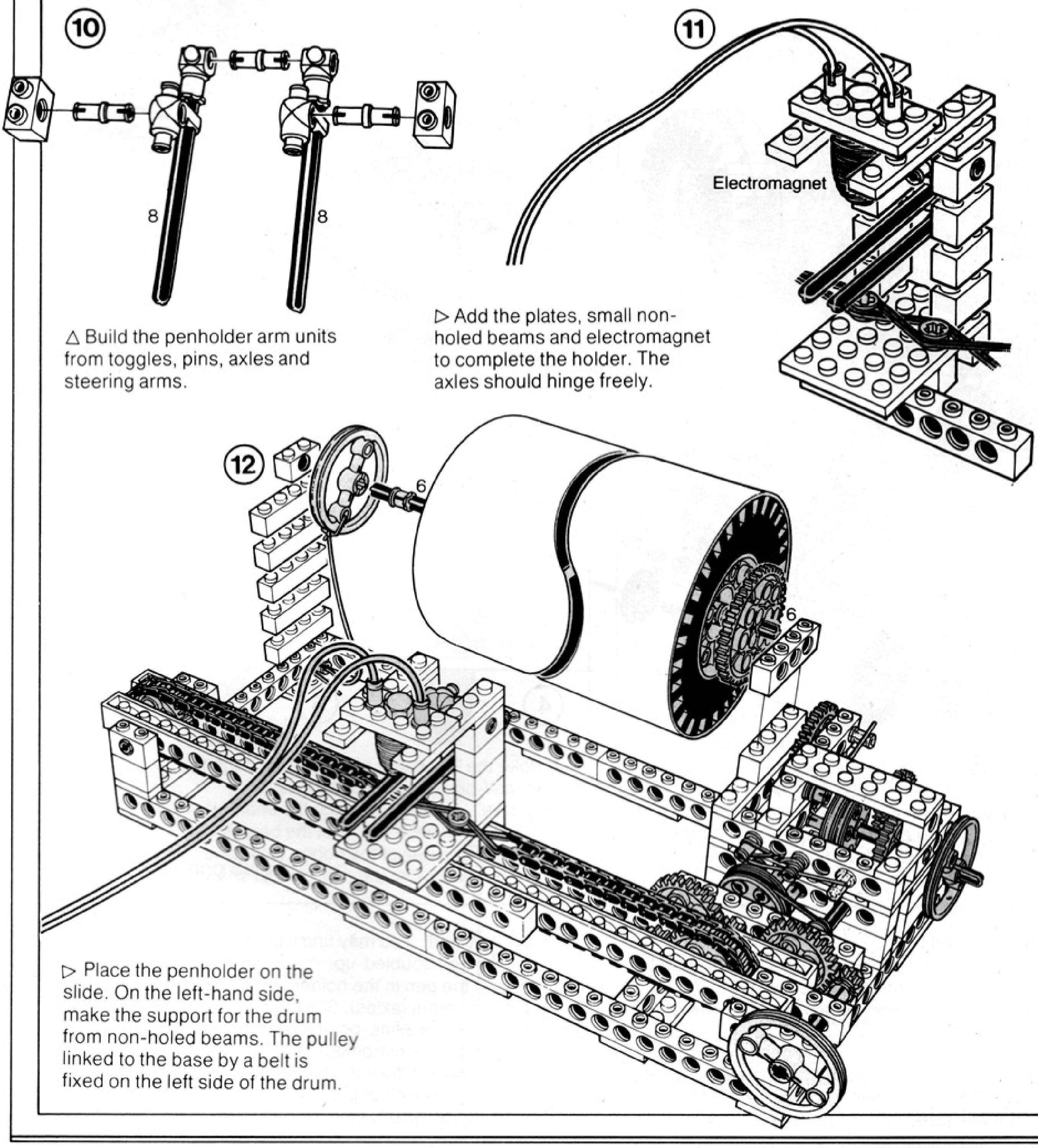
▷ Make up the slide that supports the penholder. The half-bushes sit on studs on the plate.

▷ Place the holder-unit slide on the model. Using a small belt, link the ends of the chain across the slide as shown.



⑨





⑩

△ Build the penholder arm units from toggles, pins, axles and steering arms.

⑪

▷ Add the plates, small non-holed beams and electromagnet to complete the holder. The axles should hinge freely.

⑫

▷ Place the penholder on the slide. On the left-hand side, make the support for the drum from non-holed beams. The pulley linked to the base by a belt is fixed on the left side of the drum.

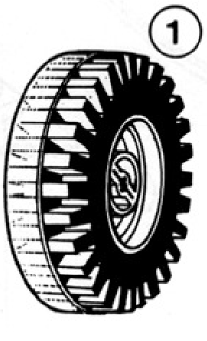
MAKING THE DRUM

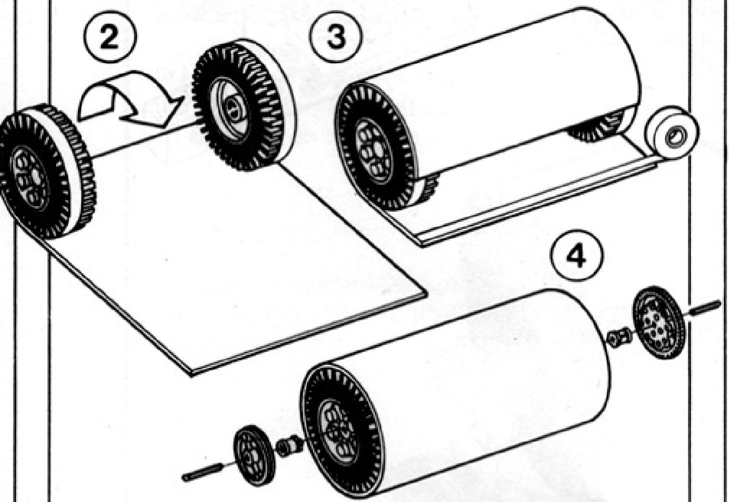
You will need 2 large LEGO wheels, a piece of card 160×280mm and double-sided sticky tape.

1 Wrap double-sided sticky tape round the outer edge of each wheel.

2 Peel off the tape backing and roll the card round the wheels. The edges of the card must align with the edges of the wheels.

3 When you have nearly finished rolling the card round, place a strip of sticky tape across the front end of the card. Peel off the backing, then stick the end down to complete the drum.





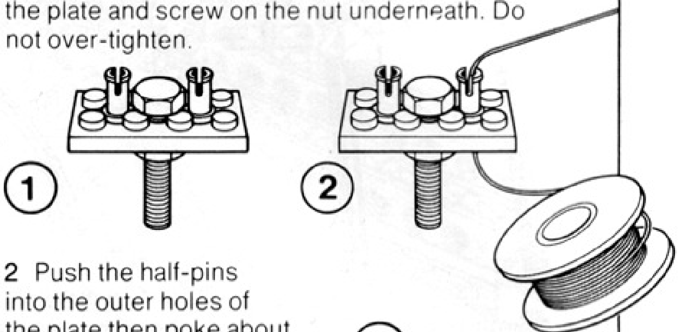
4 Into each wheel, push a 6-length axle bearing a bush and a 40-tooth cog wheel. The axles fit through the end beam-stacks on the plotter.

Wrap a sheet of drawing paper round the drum and stick down the ends using Blu-Tack. When you have finished your drawing, simply remove the paper and replace it with a new sheet.

MAKING THE MAGNET

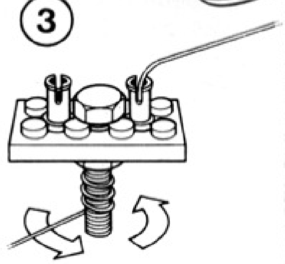
You will need a 2×4 holed plate, 2 half-pins, 1 steel M5 25mm bolt with nut, and some 34 s.w.g. enamelled wire.

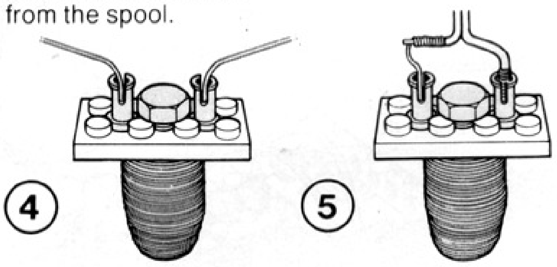
1 Poke the bolt through the central hole of the plate and screw on the nut underneath. Do not over-tighten.



2 Push the half-pins into the outer holes of the plate then poke about 10cm of wire through one of the half-pins.

3 Wind the wire round the bolt about 800 turns. (Write down the numbers 1 to 8 and each time you reach 100, cross off the next number.) Cut the wire from the spool.





4 Poke the loose end of the wire through the other half-pin. Cut the ends to leave long 'tails'. 5 Gently scratch the enamel off the ends of each tail, twist the bare ends with the bare wires of the ribbon cable and poke the 'plugs' into the half-pins as shown.

Operating hints and tips

1 The pen should rest on the paper so that the paper fastener is about 1mm from the electromagnet. Adjust the gap between fastener and magnet by sliding the pen in and out of the holder.

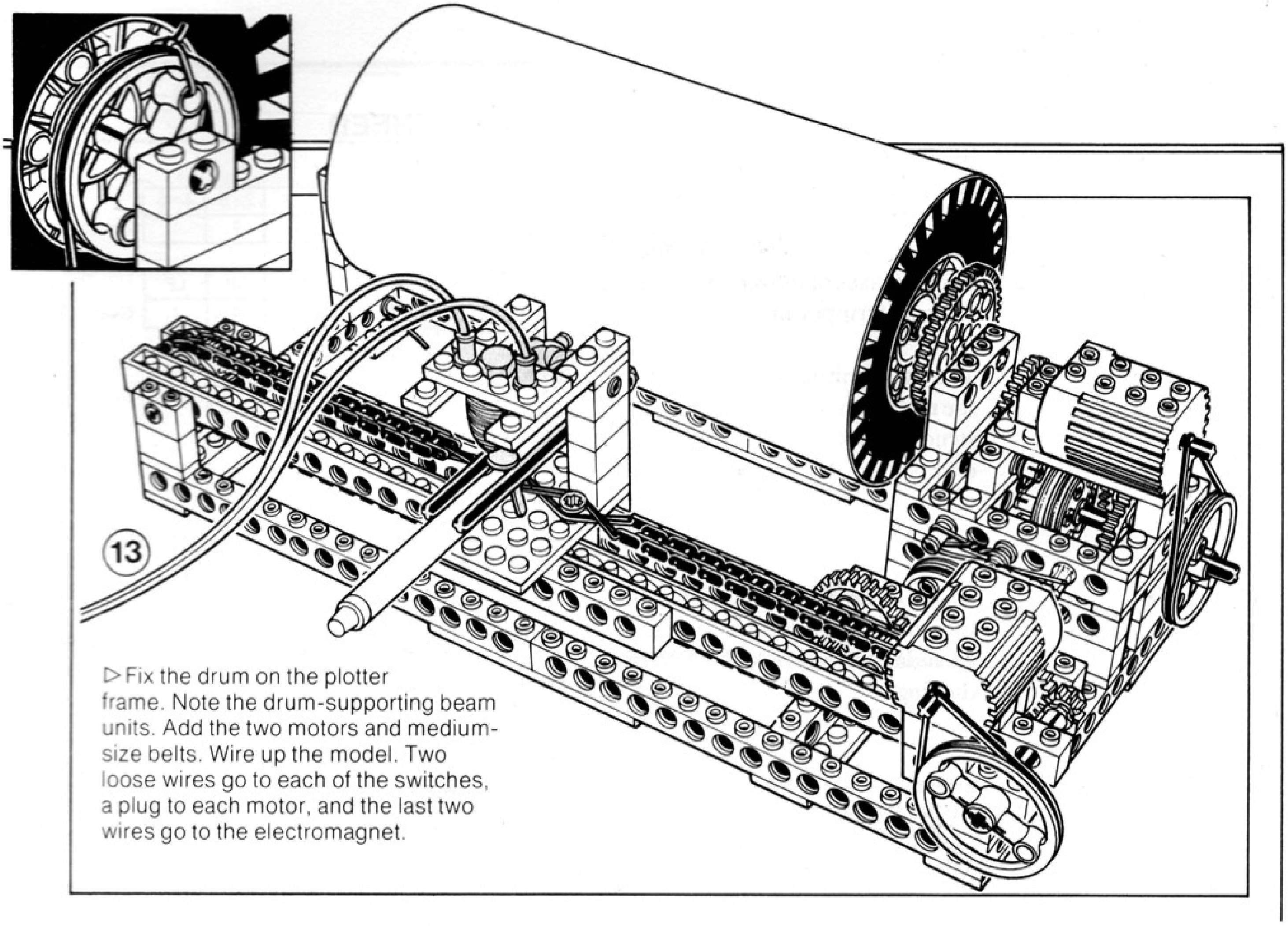
2 To keep the electromagnet wiring neat, make a junction on top of the front motor to connect the magnet to the ribbon cable. Place two small smooth plates on the motor and, with double-sided sticky tape, attach a 2-way piece of terminal block (obtainable from an electrical shop). See main photo

3 The belt used in the chain drive gears (step 3) should be small and very tight. You may find it best to use a medium-size belt doubled-up.

4 To support the pen in the holder, wrap a belt round the pen and holder arms (axles). Slip the paper fastener between them. Use a fine-point felt tip pen.

5 Do not leave the electromagnet on for too long. It will get quite hot and will drain the batteries very quickly.

6 You can use any kind of paper to draw on but for best results use a good quality cartridge paper such as typing paper.



▷ Fix the drum on the plotter frame. Note the drum-supporting beam units. Add the two motors and medium-size belts. Wire up the model. Two loose wires go to each of the switches, a plug to each motor, and the last two wires go to the electromagnet.

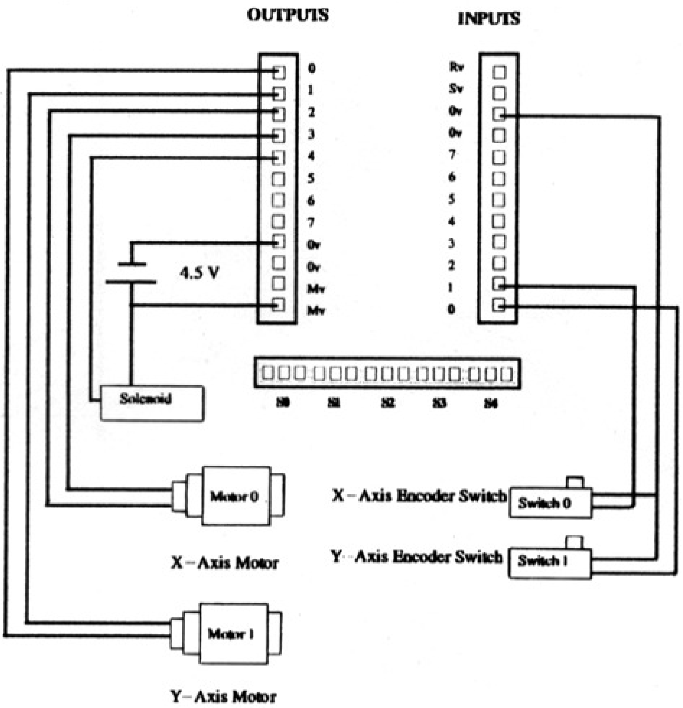

OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
Mv
Mv
Rv
Sv
0v
0v
7
6
5
4
3
2
1
0
4.5 V
Solenoid
S0
S1
S2
S3
S4
Motor 0
X – Axis Motor
Motor 1
Y – Axis Motor
X – Axis Encoder Switch
Switch 0
Y – Axis Encoder Switch
Switch 1

Figure 2.8 Wiring diagram for Drum Plotter

2.5. Maxi-arm

We've called this project the Maxi-arm because it is more versatile than the 'mini-arm' at the start of this chapter. It uses three motors to control two independent axes of movement plus a versatile gripper unit.

The base unit is the same as for the mini-arm (so refer to the construction notes for that project) except that it needs to be supported on two baseboards for extra stability.

As with the mini-arm, you may find that self-adhesive draught excluder will improve the grip of the fingers.

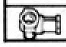
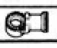
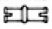
WHAT YOU NEED

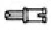
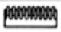
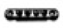
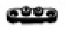
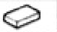
1×2	1×3	1×4	1×8	2×4	2×8	4×6	
7	6	7	2	9	7	1	Plates

8	14C	24	24C	40	SP	LP	
8	4	4	1	1	4	1	Gears

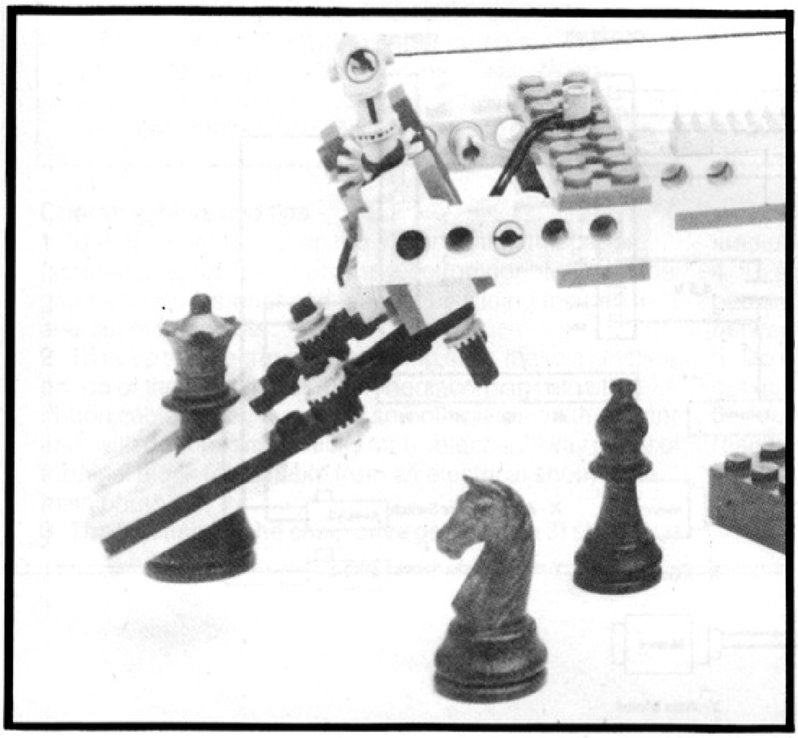
2	4	6	8	12	16	
9	8	12	10	1	2	Beams

2	4	6	8	10	
3	4	4	4	1	Axles

1	1	8	12	2	25
					

1	6	2	2	2
				

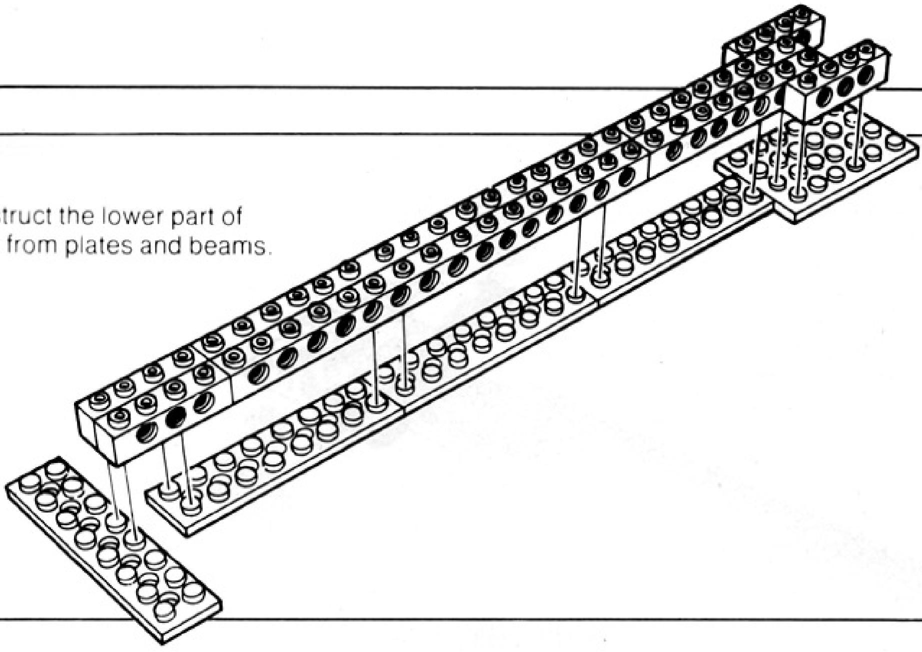
3 motors, 2 battery boxes, 6 small belts, 2 medium belts, 5 plugs, 2 10×20 base plates, short length of draught-excluder tape, 100cm of 12-way ribbon cable, 3 2.5cm squares of cooking foil, paper clips, 30cm length of string

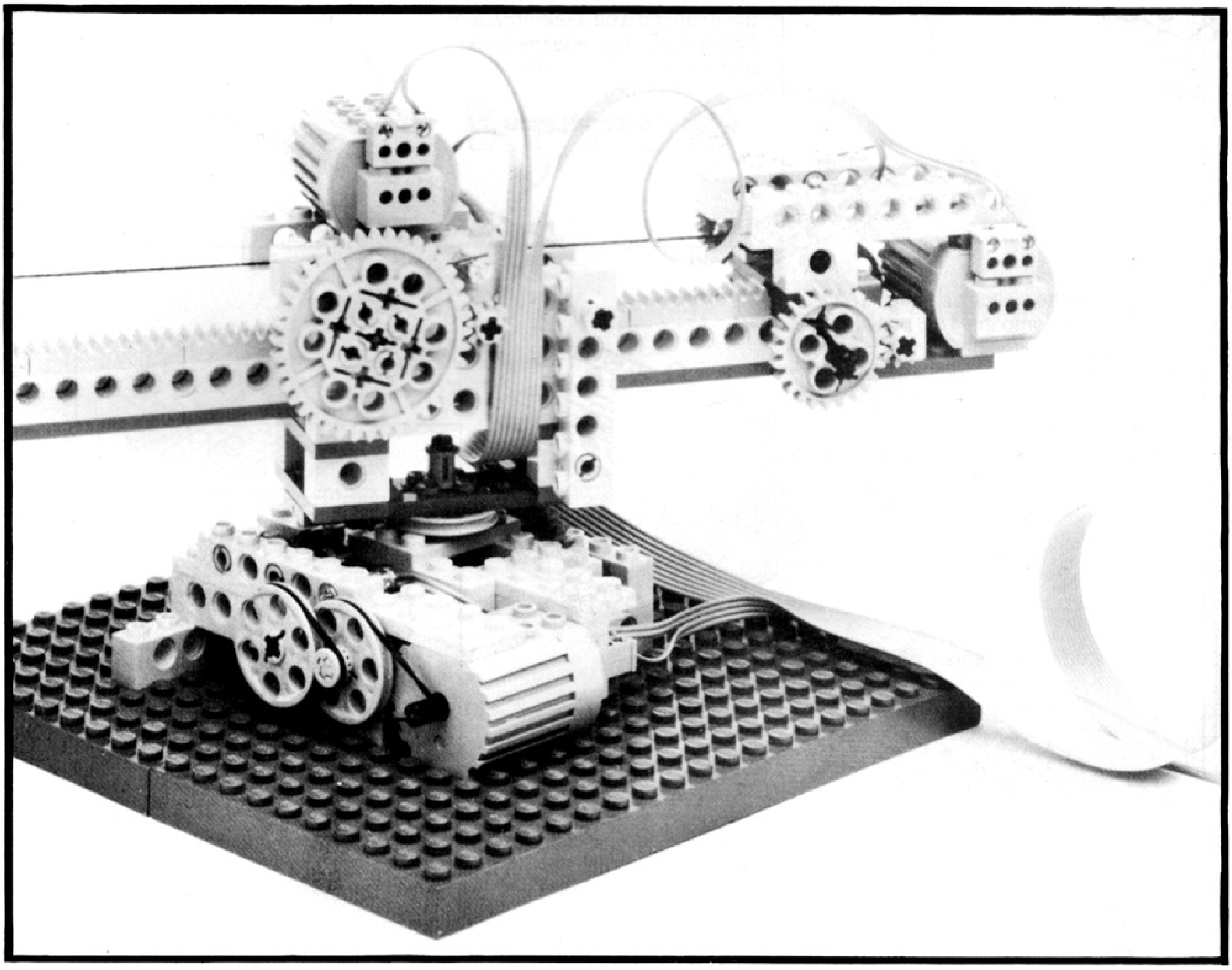


With this arm, you could move the pieces in a game of chess if pieces and board are quite small.

THE ARM

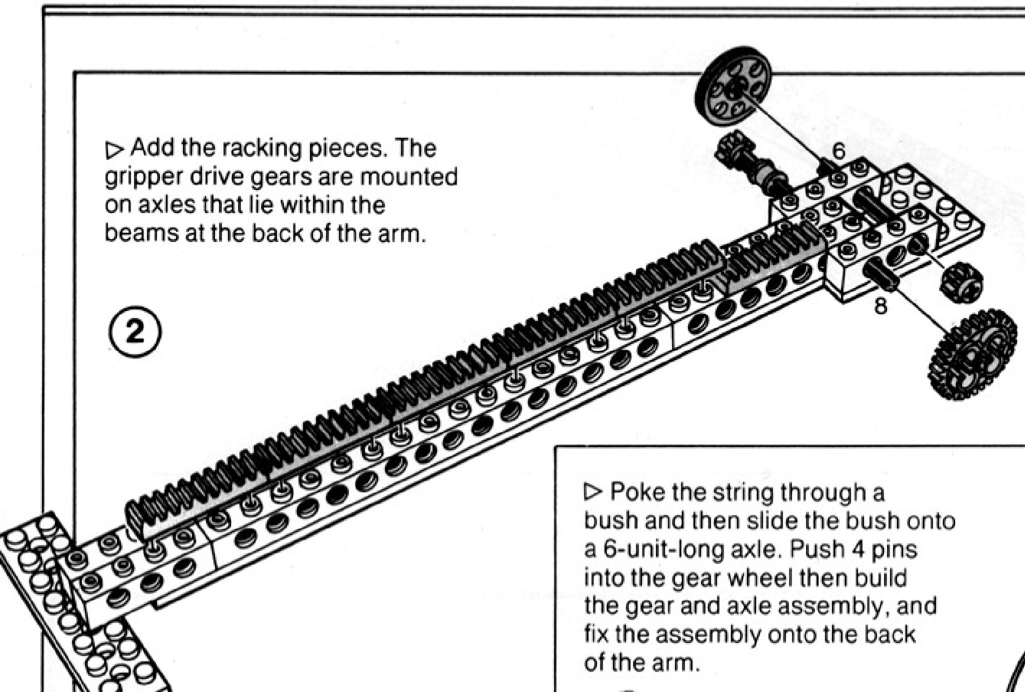
① ▷ Construct the lower part of the arm from plates and beams.





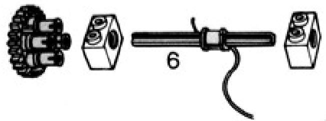
▷ Add the racking pieces. The gripper drive gears are mounted on axles that lie within the beams at the back of the arm.

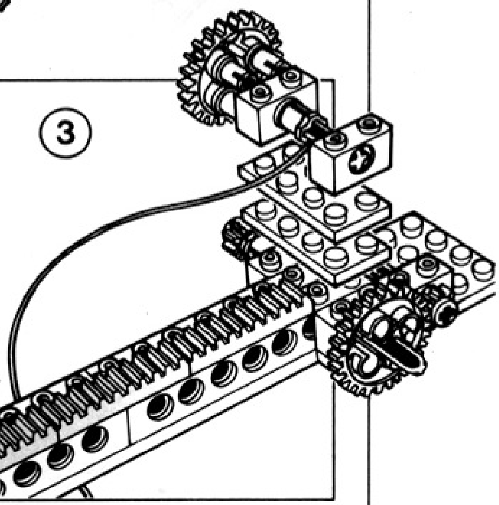
②



▷ Poke the string through a bush and then slide the bush onto a 6-unit-long axle. Push 4 pins into the gear wheel then build the gear and axle assembly, and fix the assembly onto the back of the arm.

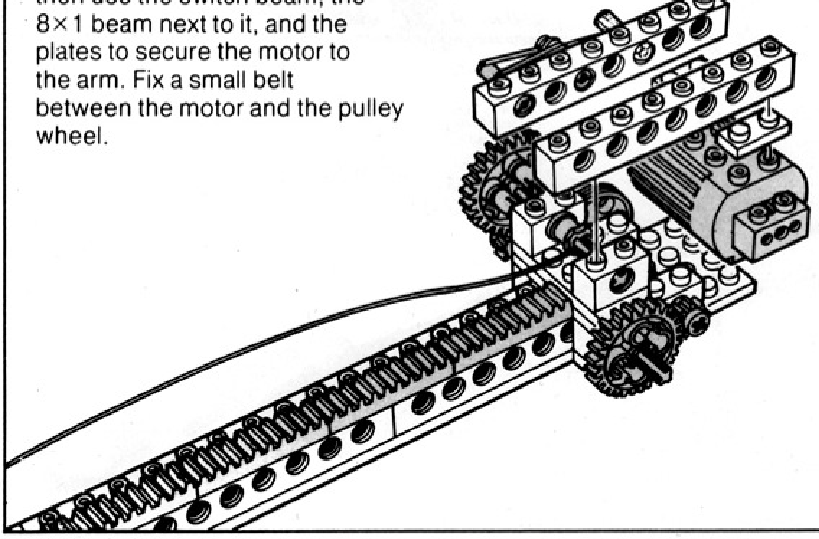
③

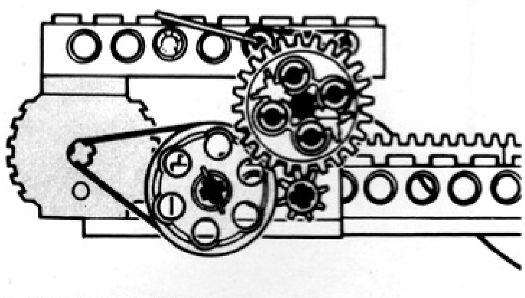




▷ Make up the switch as shown. Fix the motor onto the 4×6 plate then use the switch beam, the 8×1 beam next to it, and the plates to secure the motor to the arm. Fix a small belt between the motor and the pulley wheel.

④



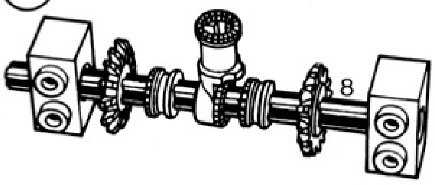


Motor/switch assembly from far side

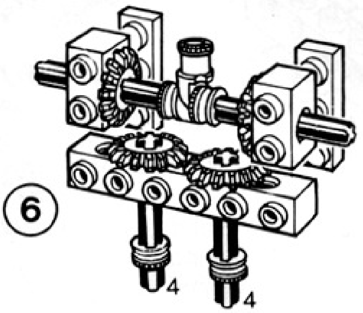
THE GRIPPER

▽Construct the main gripper axle unit ensuring that the two 14-cog gears are facing opposite directions as shown.

⑤

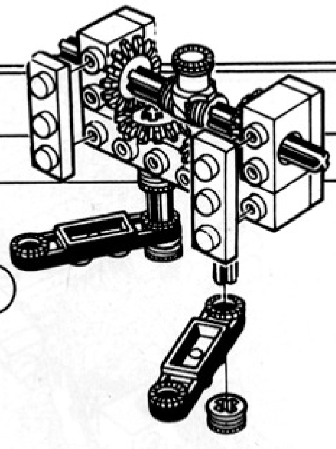


⑥

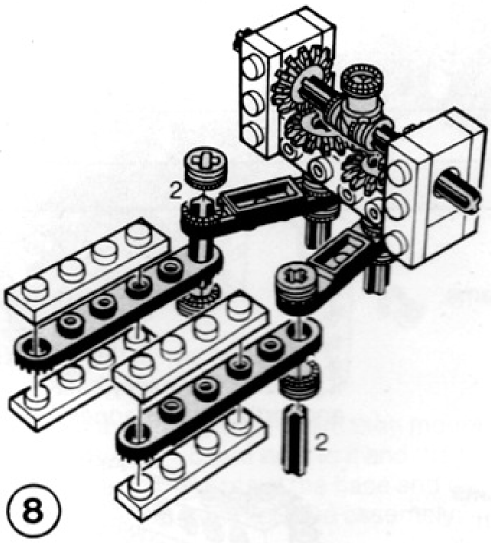


△ Add the 6-length beam, short axles, and two more 14-cog gears. Slide half-bushes part of the way up each axle.

⑦

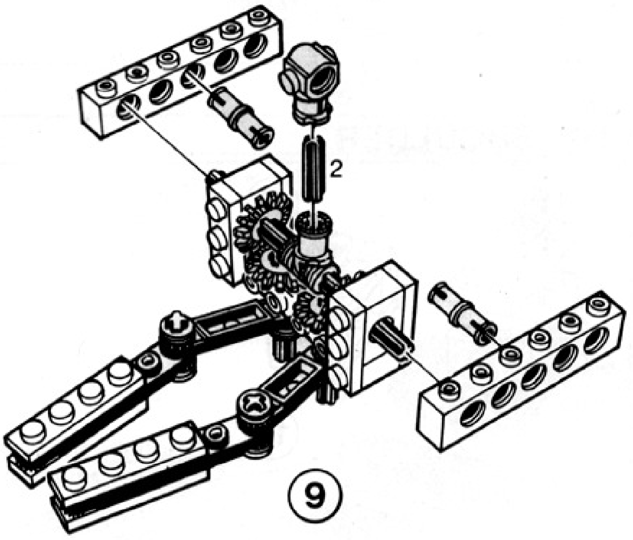


△On the 4-length axles, mount the 4-length steering plates, setting them in a V-shape. Lock them onto the axles using half-bushes. When the steering plates are moved in and out, the half-toggle on the top should move back and forth.



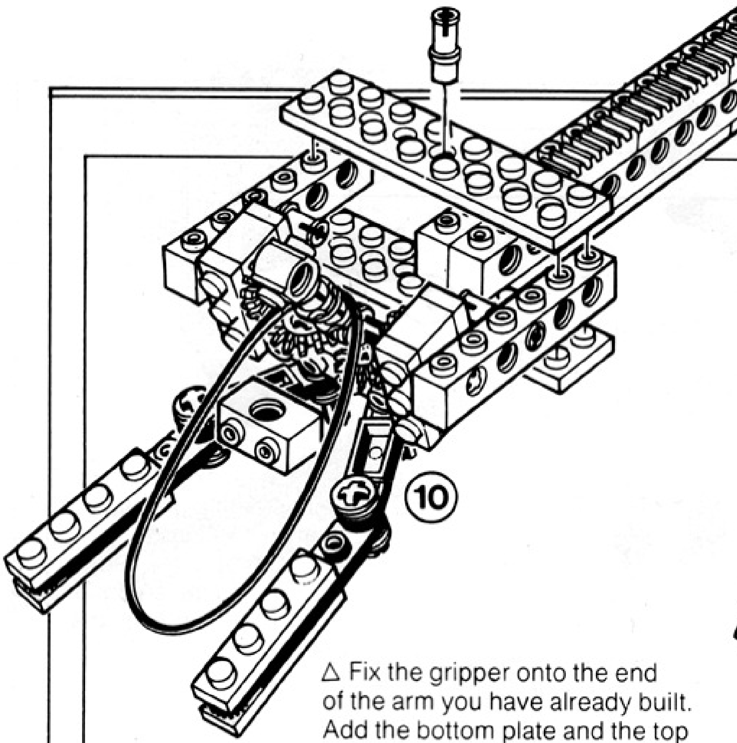
⑧

△Push 2-length axles through the ends of the short steering plates then complete the 'fingers' with 6-length steering plates and 4×1 plates. Use half-bushes on the ends of the axles to lock the steering plates together.

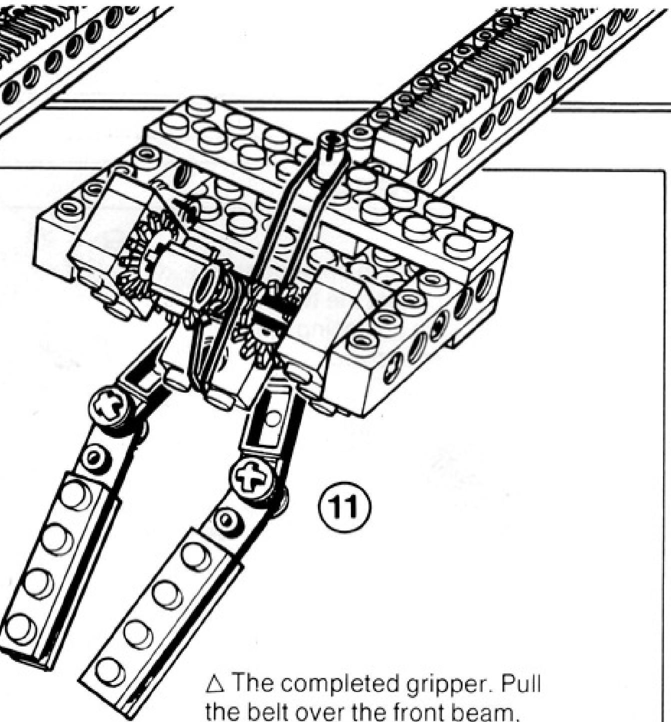


⑨

△Add the 6×1 side beams bearing pins and fit a short axle with a toggle in the top, central, half-toggle.

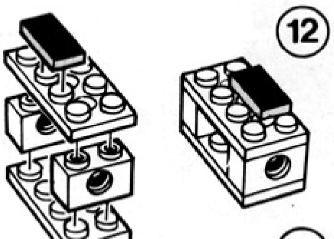


△ Fix the gripper onto the end of the arm you have already built. Add the bottom plate and the top plate bearing a half-pin. At the front, add the 2×1 beam. Hook a medium-size belt over the toggle unit.

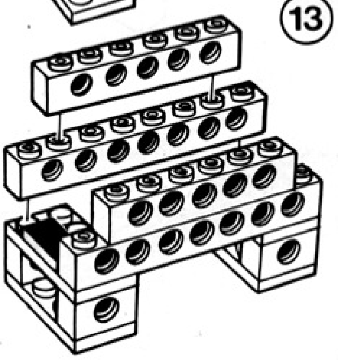


△ The completed gripper. Pull the belt over the front beam, underneath, then up onto the half-bush. The gripper should spring open in a downward position.

THE SHOULDER

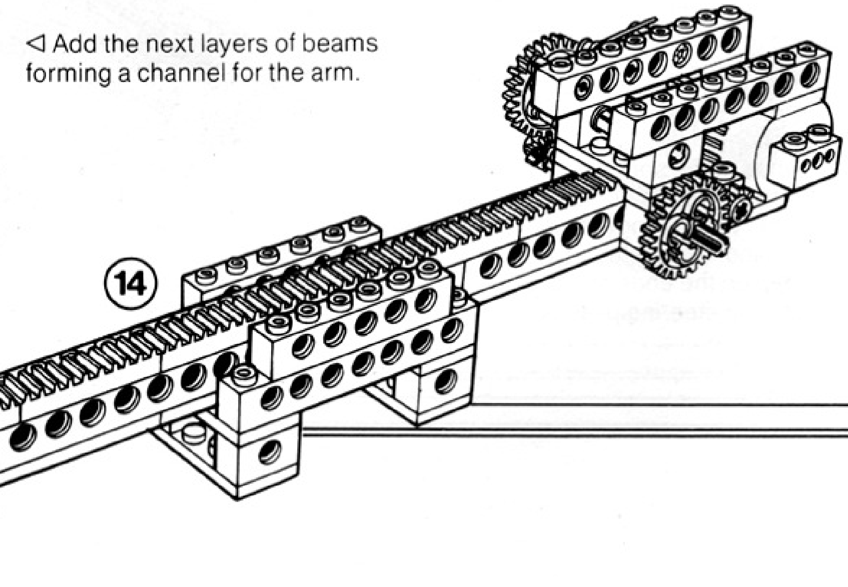


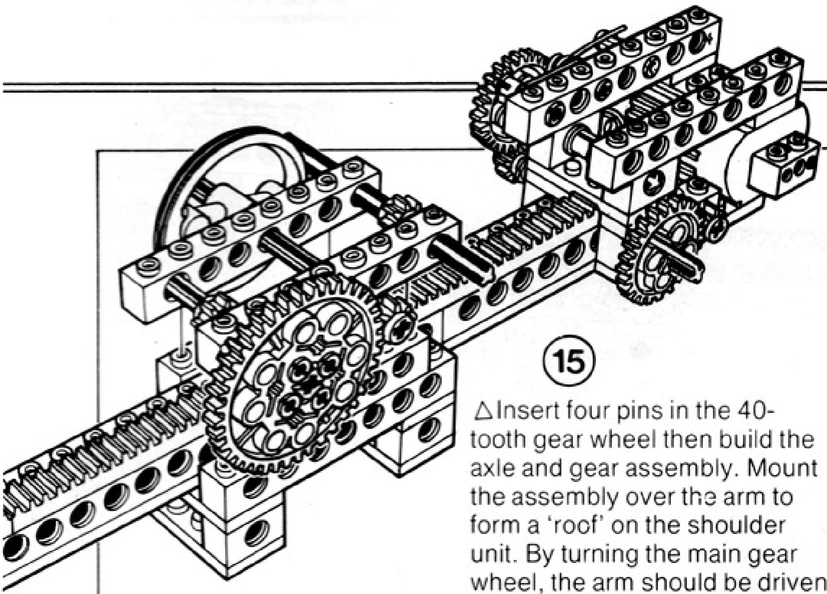
◁ Make up the base of the shoulder with plates and beams.



◁ Add the next layers of beams forming a channel for the arm.

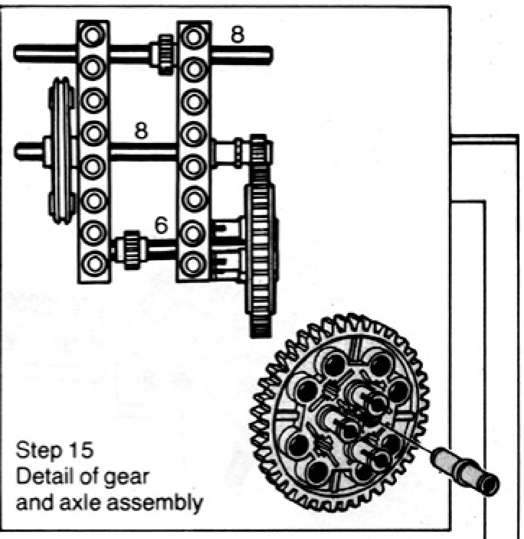
▽ Place the arm in the shoulder. It should slide easily back and forth in the channel.



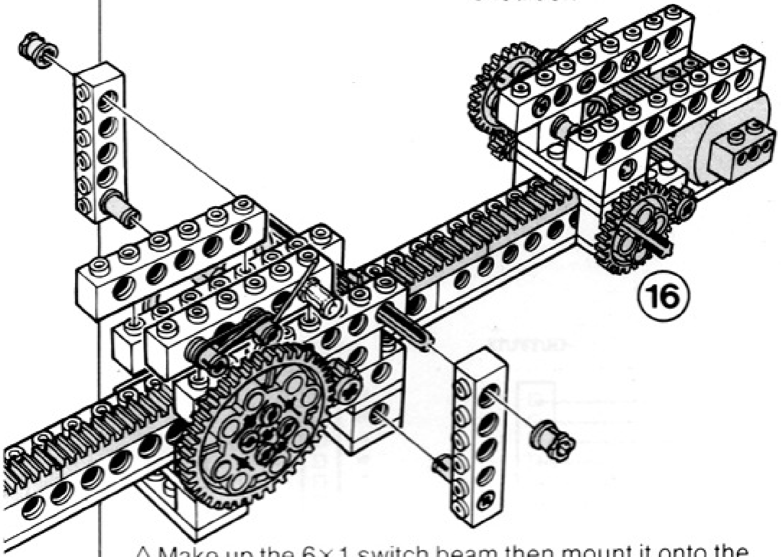


⑮

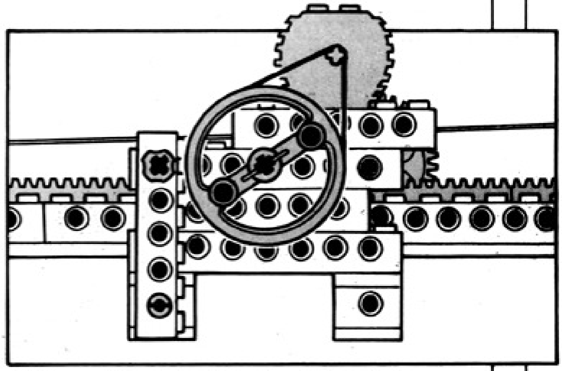
△Insert four pins in the 40-tooth gear wheel then build the axle and gear assembly. Mount the assembly over the arm to form a 'roof' on the shoulder unit. By turning the main gear wheel, the arm should be driven smoothly in and out of the shoulder.



Step 15
Detail of gear
and axle assembly



⑯

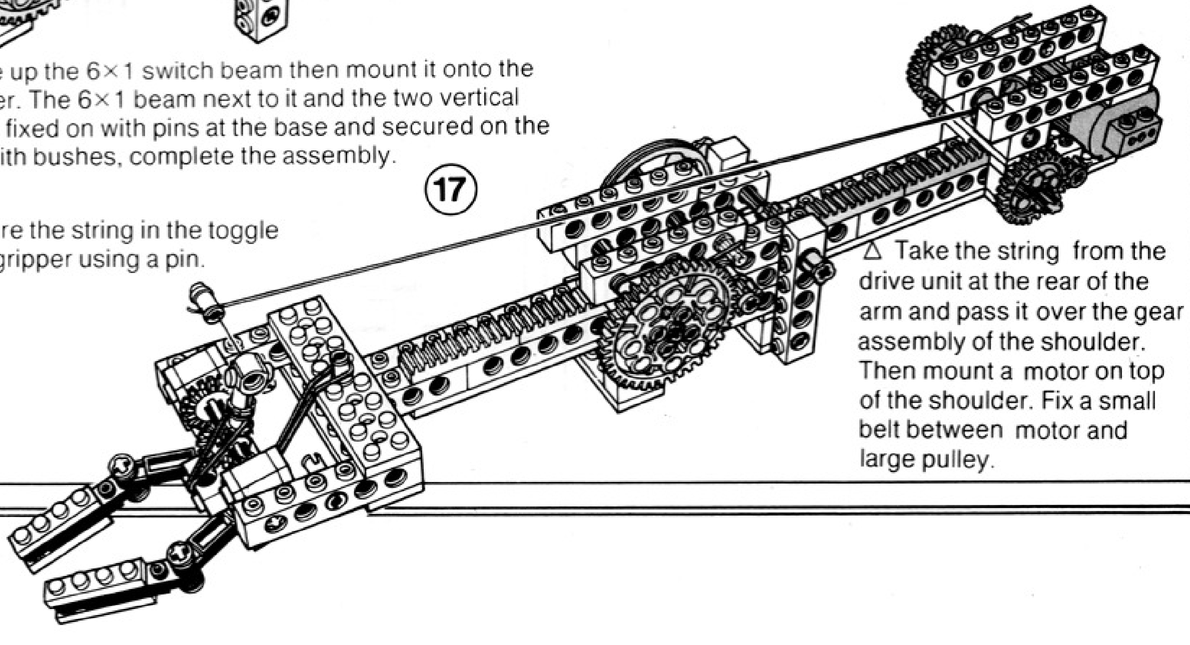


△View of the shoulder from the side.

△ Make up the 6×1 switch beam then mount it onto the shoulder. The 6×1 beam next to it and the two vertical beams, fixed on with pins at the base and secured on the axles with bushes, complete the assembly.

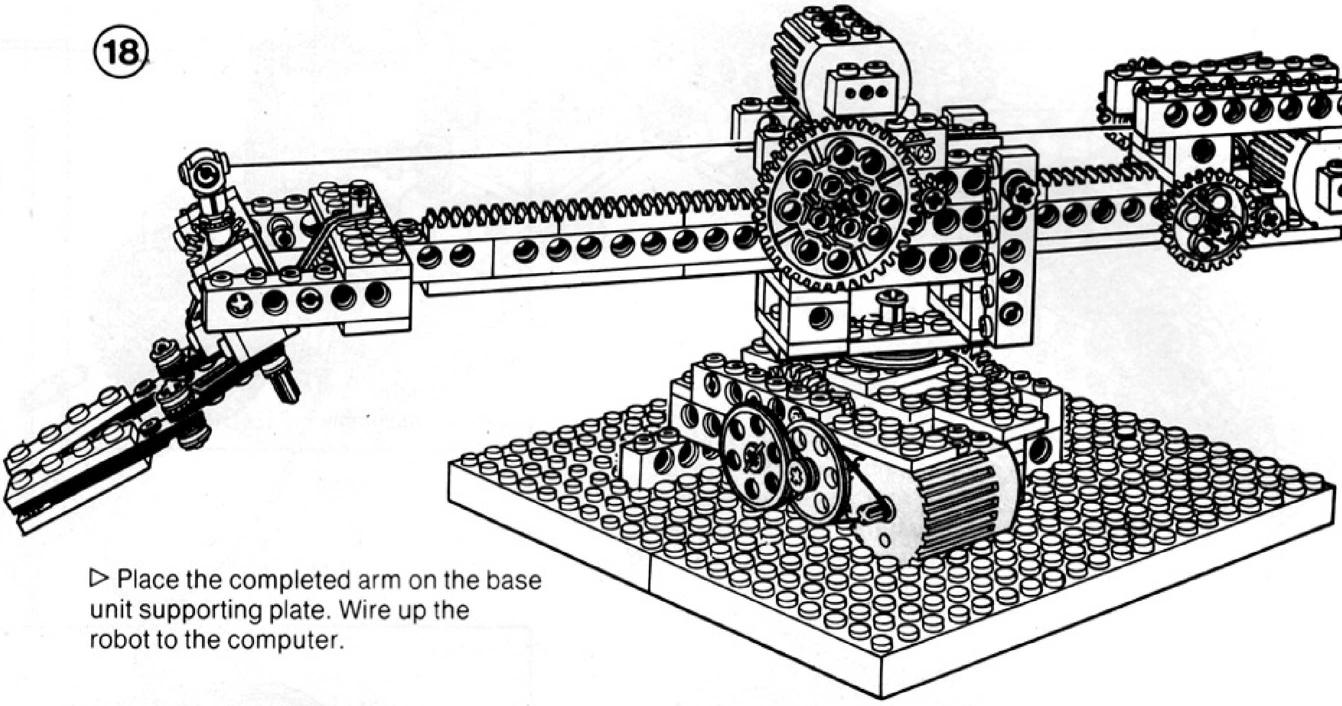
⑰

▽ Secure the string in the toggle on the gripper using a pin.



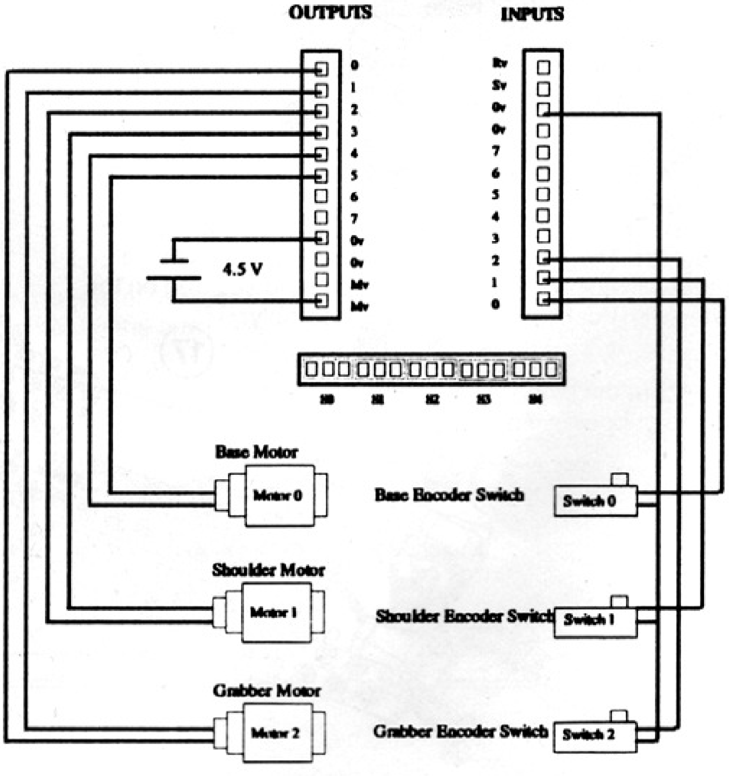
△ Take the string from the drive unit at the rear of the arm and pass it over the gear assembly of the shoulder. Then mount a motor on top of the shoulder. Fix a small belt between motor and large pulley.

⑱



▷ Place the completed arm on the base unit supporting plate. Wire up the robot to the computer.

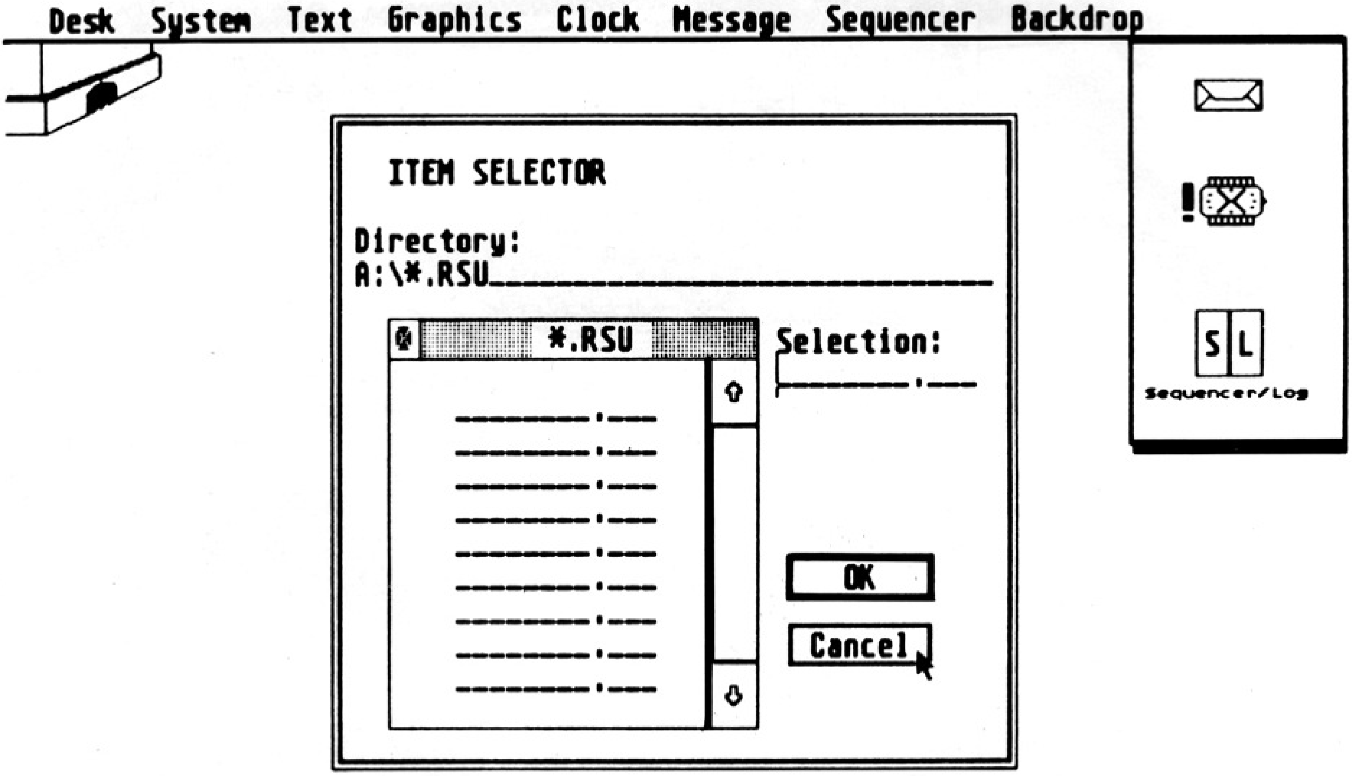
Figure 2.9 Wiring diagram for Maxi-Arm


OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
9v
9v
4.5 V
9v
5v
0v
0v
7
6
5
4
3
2
1
0
M0
M1
M2
M3
M4
Base Motor
Motor 0
Base Encoder Switch
Switch 0
Shoulder Motor
Motor 1
Shoulder Encoder Switch
Switch 1
Grabber Motor
Motor 2
Grabber Encoder Switch
Switch 2

3. Software Guide

3.1. Getting started

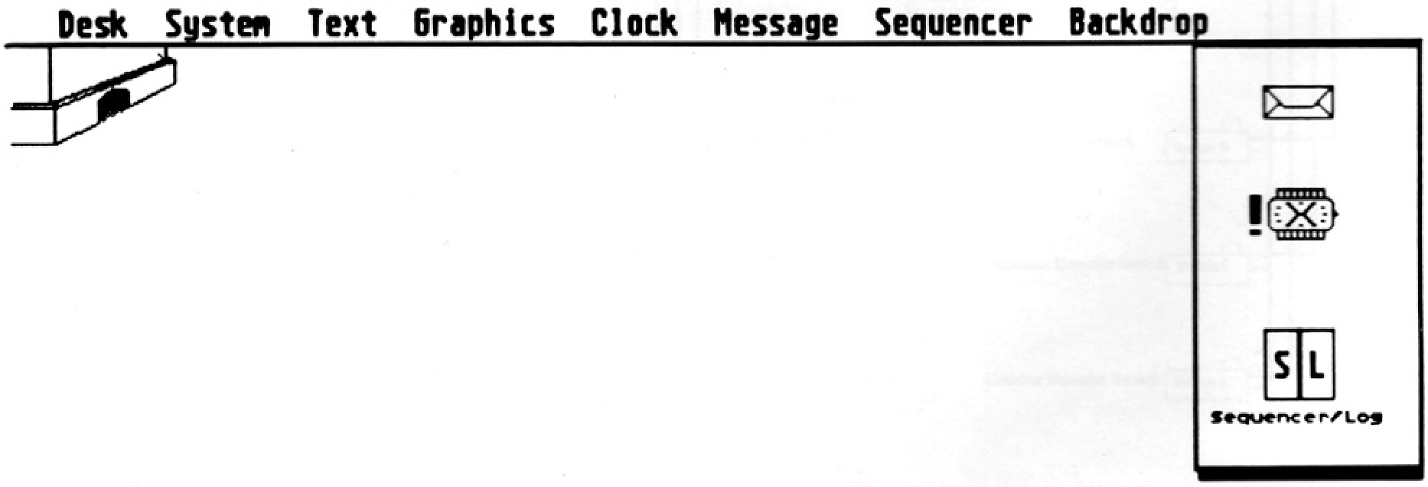
Insert the Robokit disk in your computer, and display its root directory. Double click on ROBOKIT.PRG to run the program. While you wait for the program to load (it takes about 40 seconds from a floppy), you'll see an hour-glass cursor icon, in which the sands of time run slowly out.


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
ITEM SELECTOR
*.RSU
Selection:
OK
Cancel
S L
Sequencer/Log

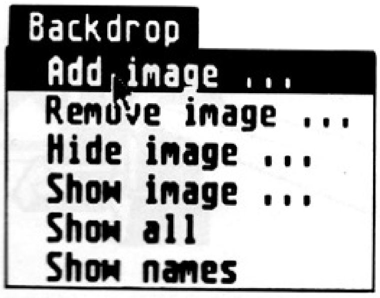
Directory:
A:\*.RSU

Eventually your screen will look like this. The file selector invites you to choose a ROBOKIT SetUp file to be loaded and run. These *.RSU files are just like the Gem DESKTOP.INF files which you'll have noticed on your Atari disks - they contain details of the screen layout and the images on it. You create them whenever you Save the Setup in ROBOKIT or Save Desktop under Gem.

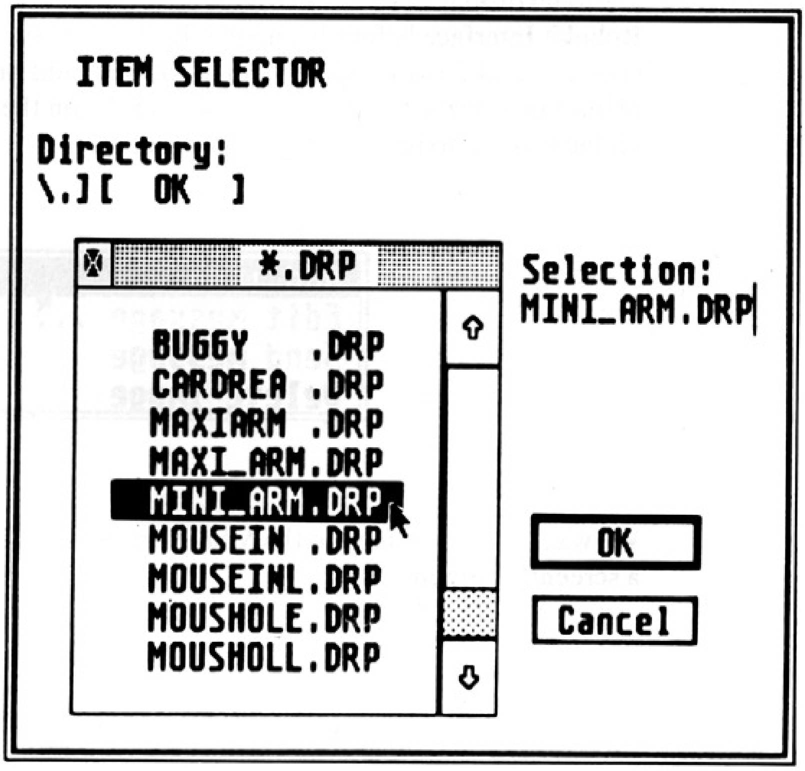
Click on the CANCEL button so that we can start from scratch with a clean screen.


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
S L
Sequencer/Log

There's the usual Gem-type menu bar, of course, and in the top- left corner is The Mousehole; if you click on it, you'll see the arrow cursor replaced by ZZZ, and a small sleeping mouse sprawls in the Mousehole. When you're actually running an application, putting the mouse to sleep like this allows Robokit to devote all the processor's power to real system demands - no time need be diverted to monitoring the position of the mouse and the state of the windows. This means that in certain complex robotic applications, putting the mouse in the mousehole will make things run slightly faster. Right now, of course, we'll need the mouse so move the 'Z's away from the Mousehole, and the arrow cursor will reappear.


Backdrop
Add image ...
Remove image ...
Hide image ...
Show image ...
Show all
Show names

Pull down the Backdrop menu. The Backdrop is the Robokit equivalent of the Gem Desktop; you can do the usual desktop things on it, such as open windows and file-selector boxes, but you can also display graphic images instead of the blank white screen. This is not only pleasing to the eye, but is an important part of Robokit's control method, as we'll shortly see. You can display pictures created by Gem Paint, Degas or NeoChrome, but they must be converted first by the MAKEDRP utility (on the Applications Disk) into a standard form. Click on Add Image and you'll see a list of the pictures already available on your disk.

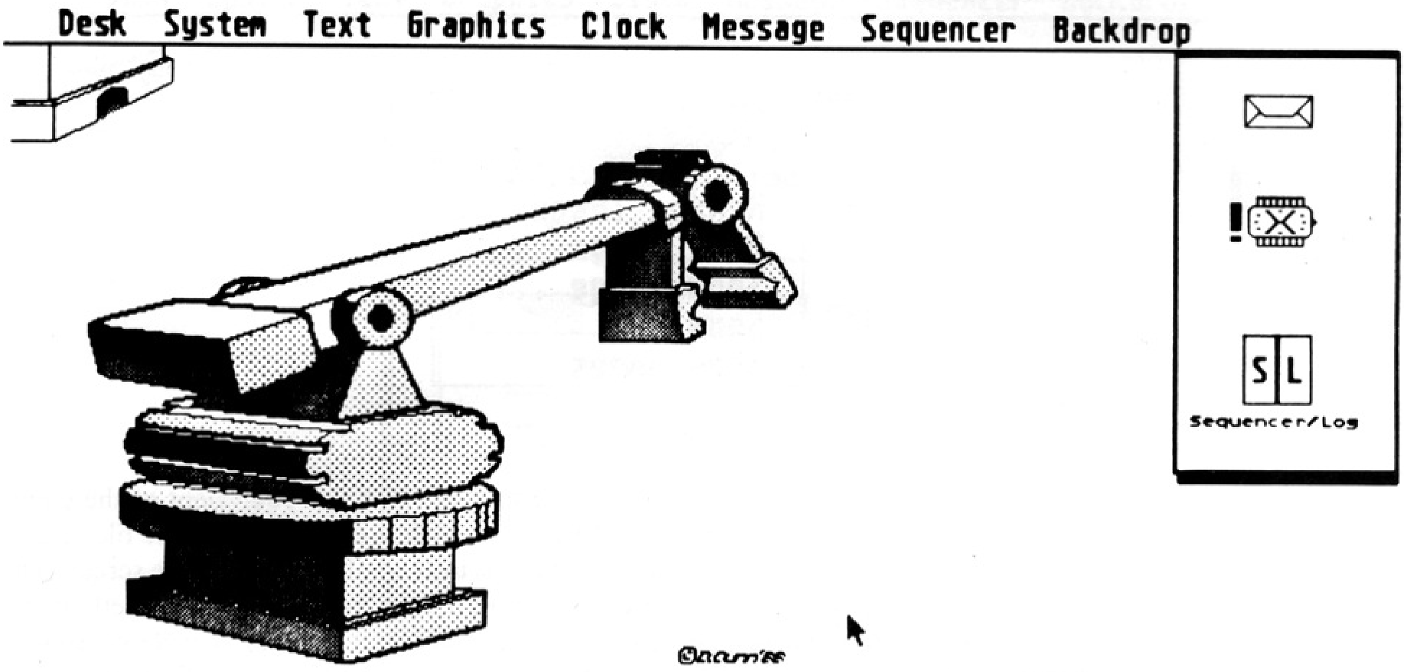

ITEM SELECTOR

Directory:
\.][OK]

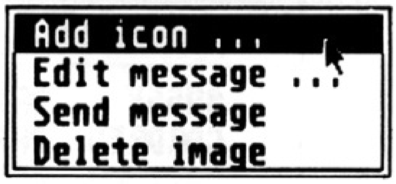
Selection:
MINI_ARM.DRP

*.DRP
BUGGY .DRP
CARDREA .DRP
MAXIARM .DRP
MAXI_ARM.DRP
MINI_ARM.DRP
MOUSEIN .DRP
MOUSEINL.DRP
MOUSHOLE.DRP
MOUSHOLL.DRP
OK
Cancel

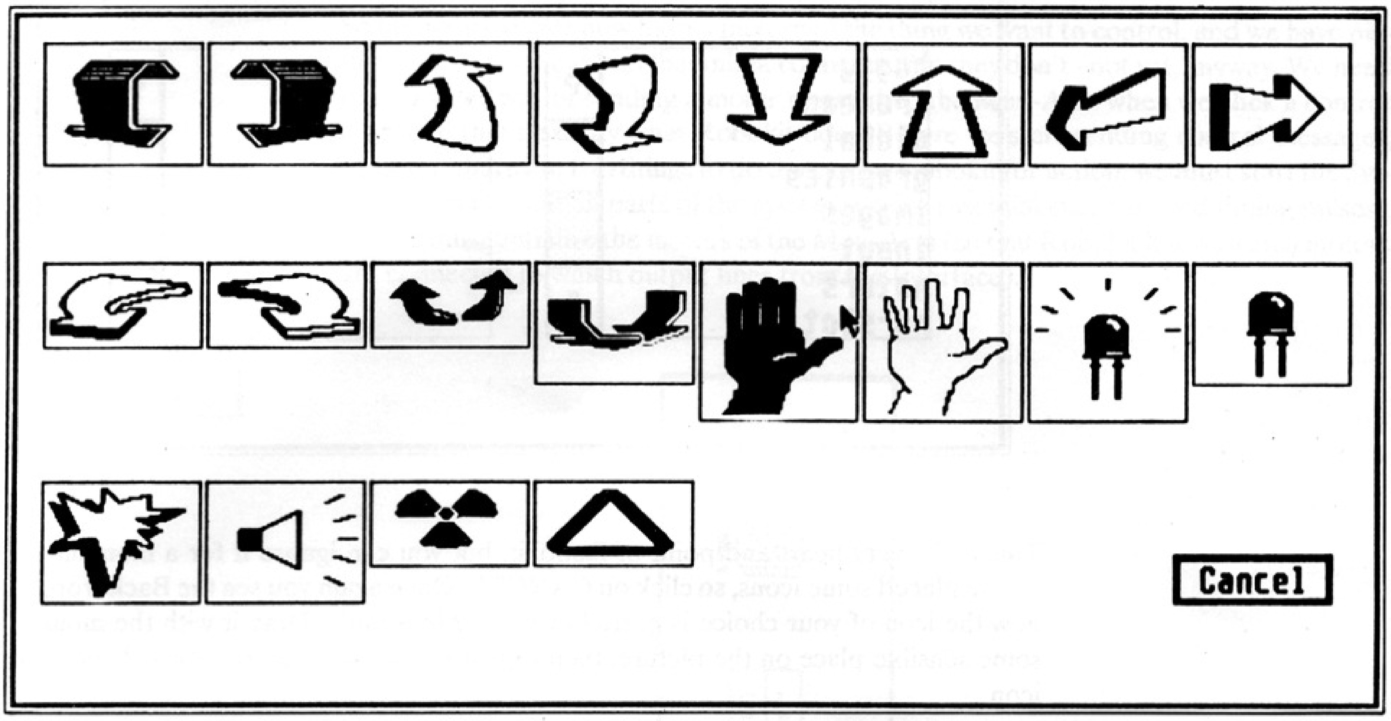
Try loading them onto the backdrop. You'll see that you can load one image after another, and that you can drag them around the screen with the mouse. The other functions on the Backdrop menu explain themselves once you've loaded some images - You can Remove them, Hide them, Show them (singly or All at once), and show their Names. You'll see that the images on the Applications Disk are all pictures of robotic devices - devices that you can build with Robokit. We'll use the Mini_Arm project in this example, so get that picture on-screen and Hide or Remove any others.


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
S L
Sequencer/Log

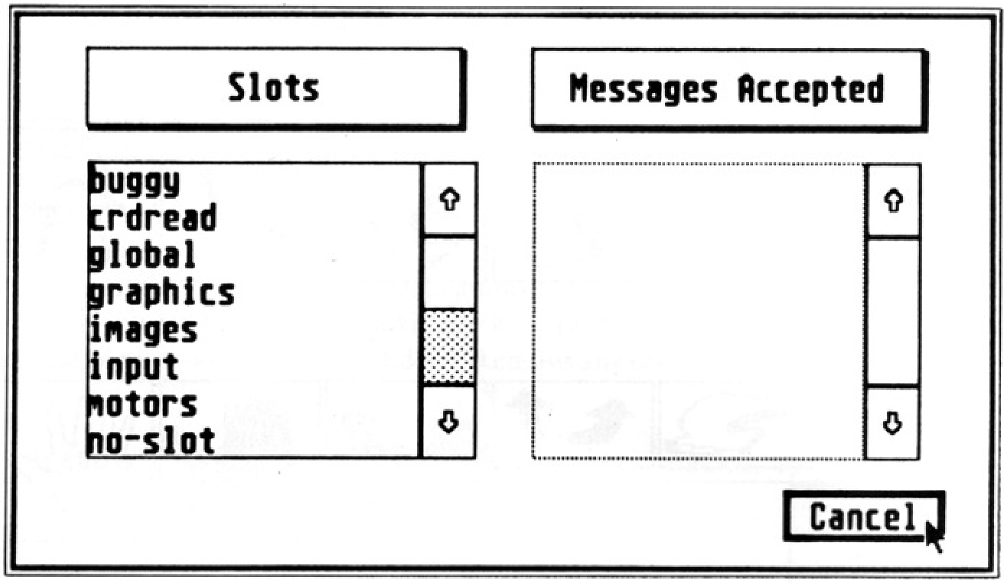
The Backdrop now shows a stylised picture of the Mini_Arm that you plugged into the Robokit Interface before beginning this session. We'll use this picture as the basis of a control panel for driving the Arm itself; we'll add push-buttons and other icons to the picture then make the Arm respond to clicks on these icons. Begin by double-clicking on the picture, to reveal a pop- up menu box.


Add icon ...
Edit message ...
Send message
Delete image

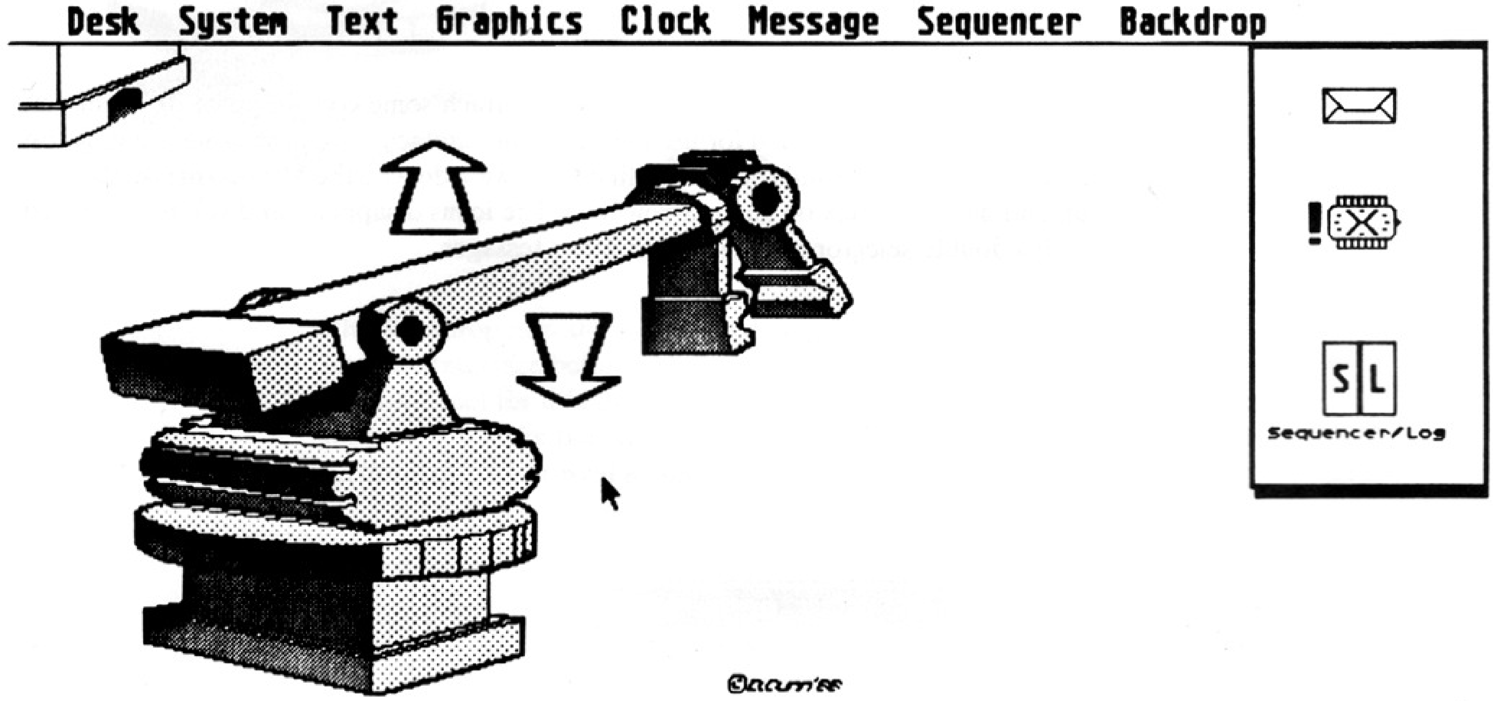
We want to Add an Icon to the Mini-Arm picture, so choose that option, and you'll see a screenful of them.



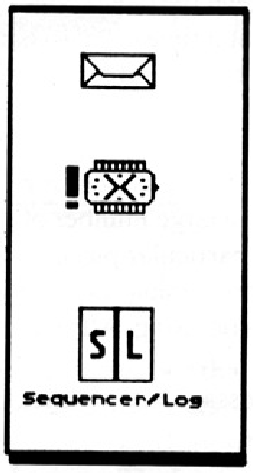
None of these icons means anything until you attach some control power to them, but they are obviously intended for use in such commonplace tasks as making things go up and down and round and round. The first thing we'll do with the Mini-Arm is make it go up and down, so click on an up-arrow icon. The icons disappear, and you're presented with a double-selector box concerned with Messages.


Slots
Messages Accepted
buggy
crdread
global
graphics
images
input
motors
no-slot
Cancel

This is the very heart and point of Robokit, but you can ignore it for a moment until you've placed some icons, so click on CANCEL. Once again you see the Backdrop, and now the icon of your choice is pasted over the Mousehole. Drag it with the mouse to some sensible place on the picture, then repeat the whole process with a down-arrow icon.

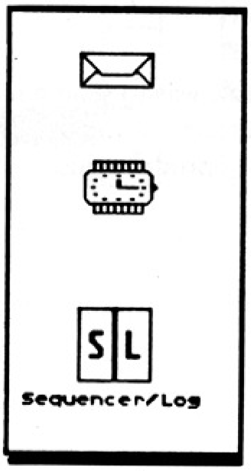

Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
S
L
Sequencer/Log

So far, so good. We now have a picture of the thing we want to control, and we have pictures of things (the icons) that might control it. But they don't - not yet, anyway. We need to find some way of sending a motor message to the Mini-Arm when we click a control icon, and that's exactly what Robokit does. Before we start sending control messages, however, there are two things to do that set up Robokit for action: we must start the System Clock (so that all parts of the system receive essential synchronised timing pulses), and we must initialise the motors of the Mini-Arm (so that Robokit knows which motors are connected to which output lines from the Interface).

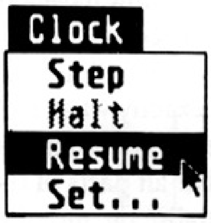


On the right of your screen is the 'control panel'. It contains an envelope (which as we shall see later is used to send messages within the system), a stopwatch (which should currently have a large X in the centre and an exclamation mark to the left), and two small boxes marked S and L - which have to do with functions called the Sequencer and Log.

Starting the System Clock is easy - just point and click on the face of the stopwatch. The hands will re-appear on the face and the exclamation mark will disappear. Clicking again will stop the clock and change the graphics back.



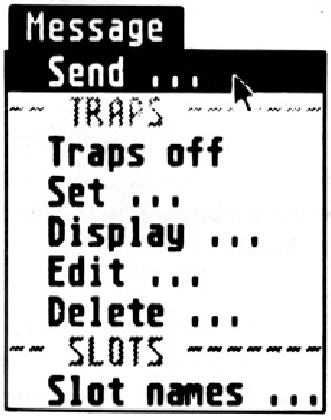
If as you build and control robots they sometimes don't move when you expect them to, first check that you have the system clock running.



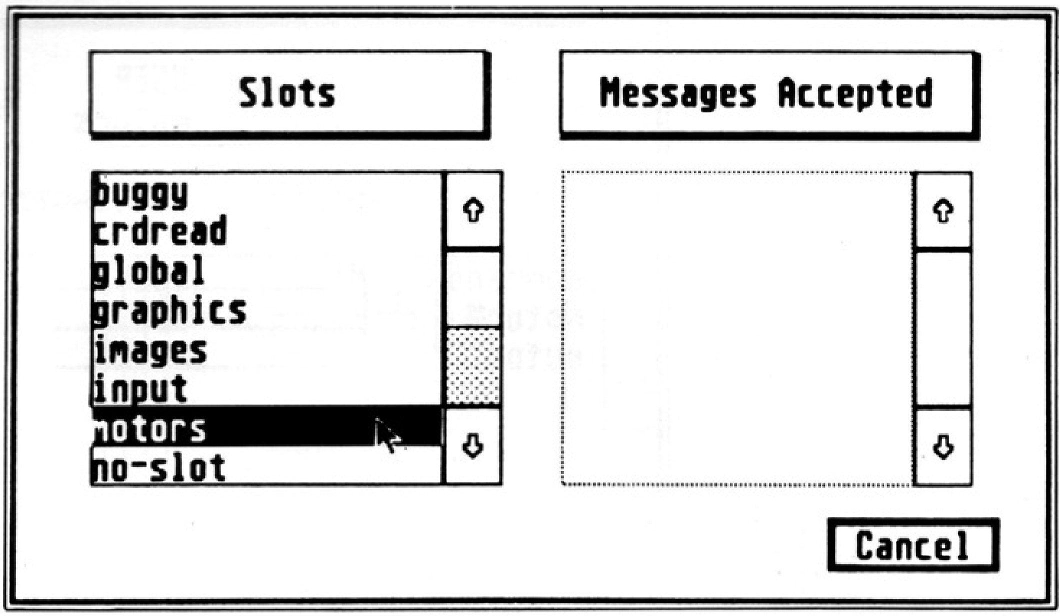
If you pull down the Clock Menu you will see the words Halt and Resume (only one will be available at a time). These have the same effect as clicking on the stopwatch to Halt and Resume the operation of the clock. The Step and Set... options are explained later.

To initialise the motors we need to send a message to the part of the software that performs all aspects of the motor control. This is called the Motor Slot. In fact, ROBOKIT is made up of a large number of slots. A slot is just a self contained software routine that looks after a particular piece of control. They are called slots because the computer has to share its processing time between them all - offering each one in turn a slot to perform some processing. All slots communicate with each other by sending messages.

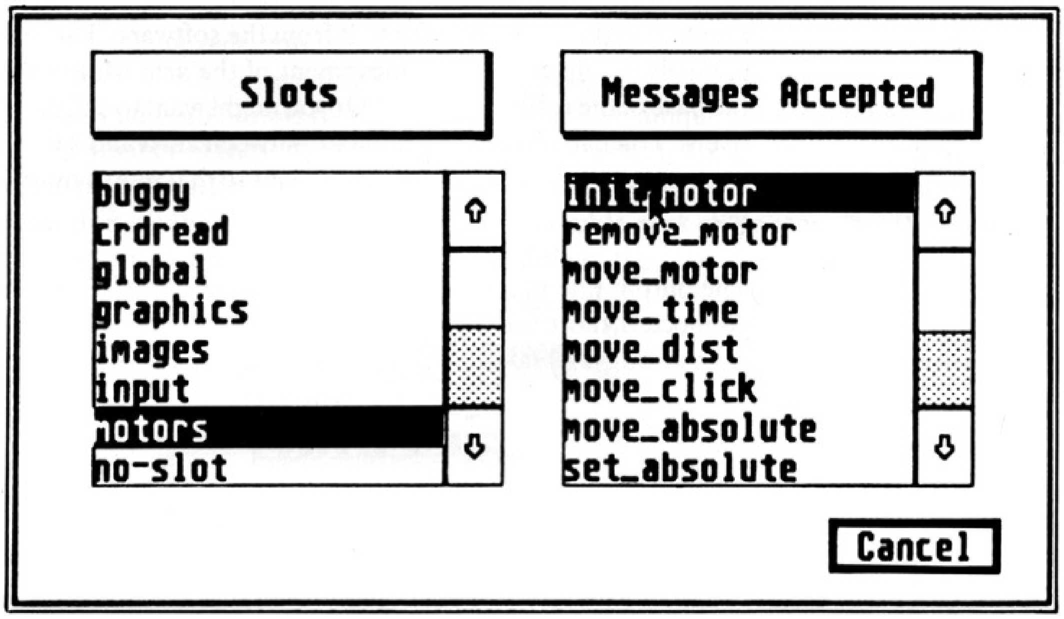
To send a message, simply click on the envelope icon in the control panel:



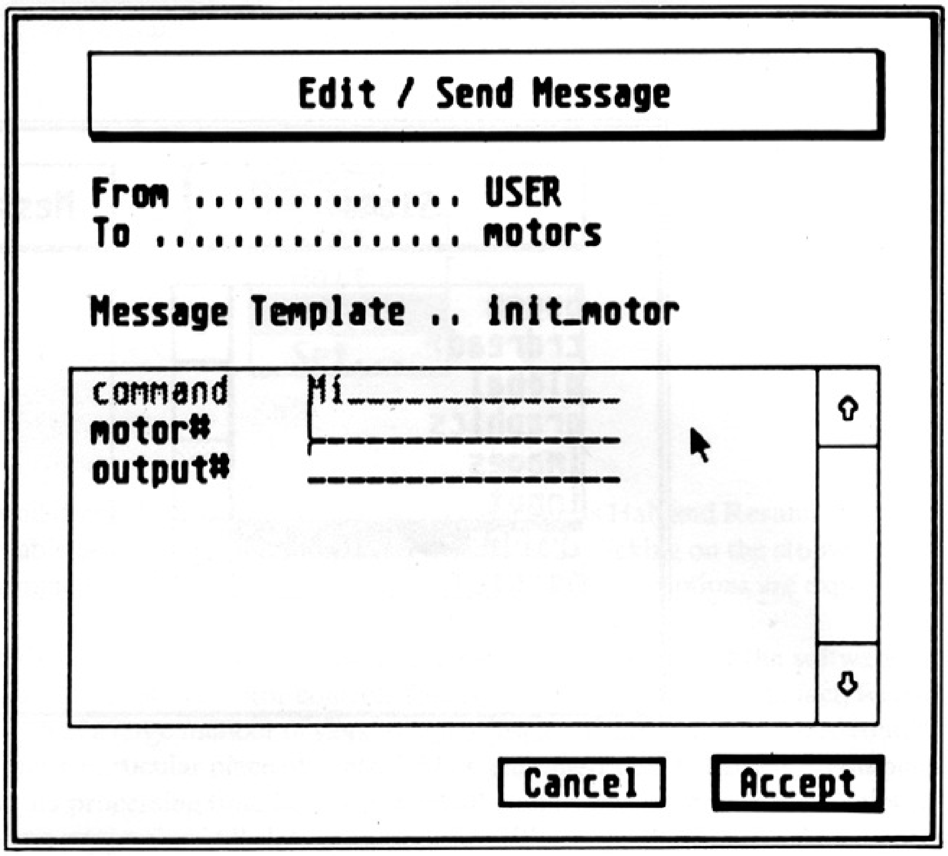
The alternative way to send a message is to click on the Send ... option in the Message menu, but we suggest you use the envelope on the control panel. Either way of sending a message will produce this:


Slots
Messages Accepted
buggy
crdread
global
graphics
images
input
motors
no-slot
Cancel

You're back now at the double-selector box that you saw when you selected an icon from the icon display. This time we'll use it. In the left-hand selector box is a list of names labelled SLOTS - these are places in the Robokit system to which you can send messages in order to make something happen. The one we want is 'Motors'.


Slots
Messages Accepted
buggy
crdread
global
graphics
images
input
motors
no-slot
init_motor
remove_motor
move_motor
move_time
move_dist
move_click
move_absolute
set_absolute
Cancel

The box on the right now shows mysterious words and phrases, the names of the pre-defined messages that you may send to Motors. The one we want is easy to guess in the circumstances, it's 'init_motor'. Click on it, and you're presented with the Message Edit Box.



Edit / Send Message

From .... USER
To .... motors

Message Template .. init_motor

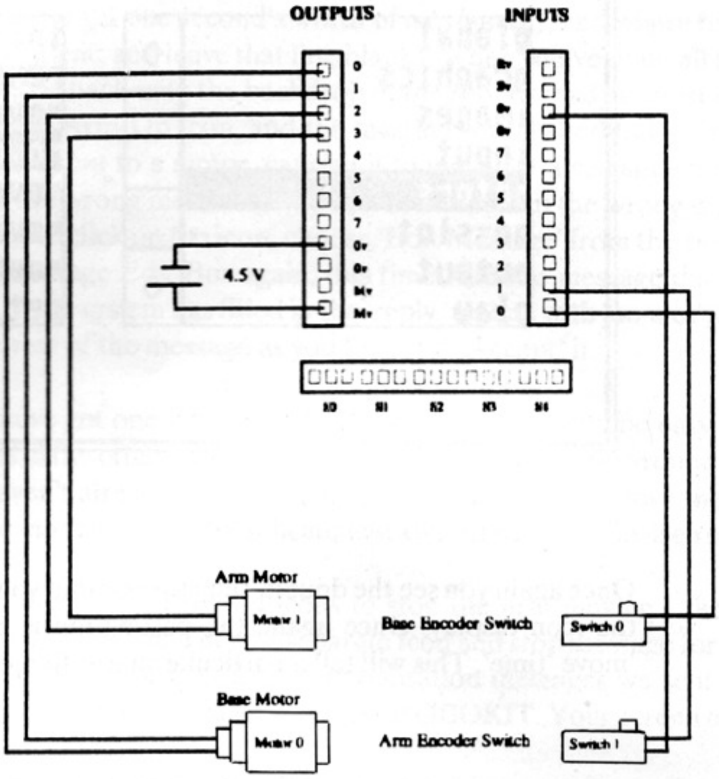
command Mi
motor#
output#

Towards the top of the Box, on a line headed ’Message Template:’, is ’init_motors’ - the name of the message we’ve chosen to Edit and Send. Above that are the names of the source (you, the USER) and destination (motors) of the message. Below that is the message itself, consisting of some text, written in grey because we can’t edit it, and some data - the number of the motor to initialise, and the number of its output line.

When you build a robot model with motors, it is necessary to give each motor a unique number so that you can refer to it from the software. Thus you might call the motor that controls the up-and- down movement of the arm Motor #1, and the motor that rotates the base of the robot Motor #2. Or you might want to call them motors #4 and #6 respectively. You can choose any number between zero and seven for the motor number, but make a note of it because other messages to that motor must include this reference number.

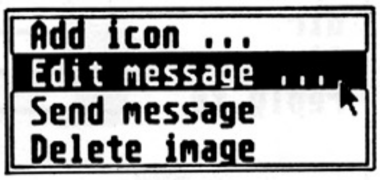
The output line number is the number of the Interface terminal to which that motor is connected; there are two wires from each motor to the Interface terminals, and the lower-numbered terminal is taken to be the output line for this message’s purpose.

Mini–Arm wiring Diagram

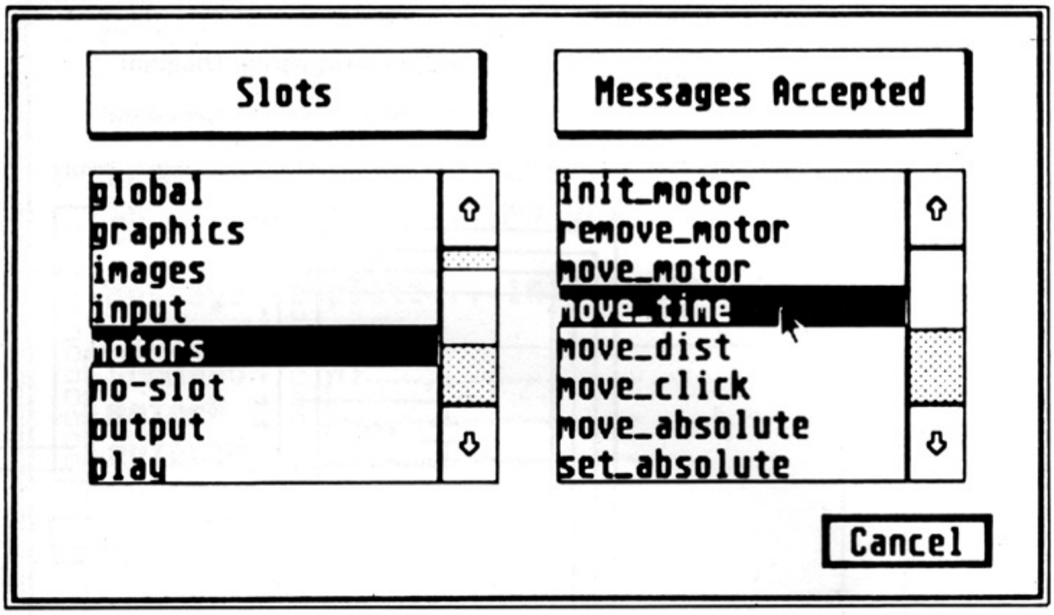

OUTPUTS
INPUTS
0
1
2
3
4
5
6
7
0v
0v
Mv
Mv
8v
8v
0v
0v
7
6
5
4
3
2
1
0
4.5 V
N0
N1
N2
N3
N4
Arm Motor
Motor 1
Base Encoder Switch
Switch 0
Base Motor
Motor 0
Arm Encoder Switch
Switch 1

Suppose, for example, that the motor controlling the arm's vertical movement, is connected to Interface terminals 2 and 3: then we might call this motor number 1, and its output number is 2.

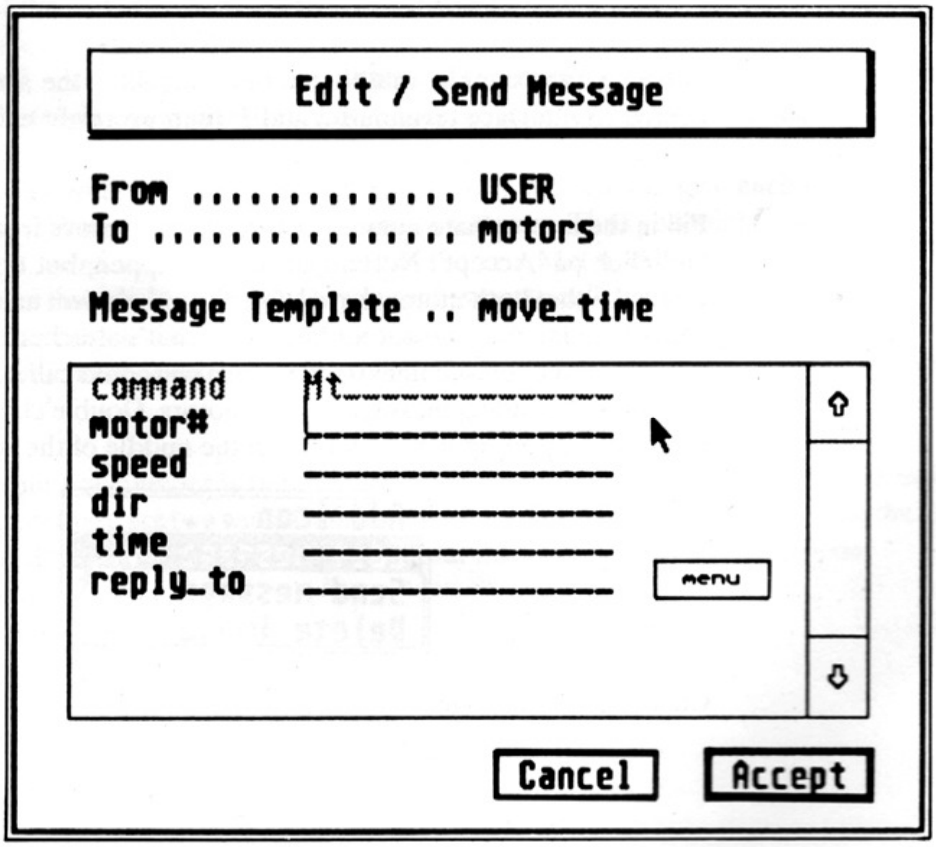
Fill in the appropriate numbers (using TAB to move from line to line in the Edit Box), and click on 'Accept'. Nothing seems to happen, but do the same thing for the other motor, so that both motors have been given their own unique motor number, associated with the number of the output line to which it's attached. For example if the base motor is connected to output lines 0 amd 1 then you could call it Motor #1 on Output #0. Now we can start sending messages to the motors. Double click on an icon - say the Arm-Up icon - and a pop-up menu appears in the middle of the screen.


Add icon ...
Edit message ...
Send message
Delete image

Choose Edit Message.


Slots
global
graphics
images
input
motors
no-slot
output
play
Messages Accepted
init_motor
remove_motor
move_motor
move_time
move_dist
move_click
move_absolute
set_absolute
Cancel

Once again you see the double-selector box that you saw when you selected an icon from the icon display. Once again, the destination is motors, but the message this time is 'move_time'. This will tell a particular motor to move for a certain length of time.


Edit / Send Message

From .... USER
To .... motors

Message Template .. move_time

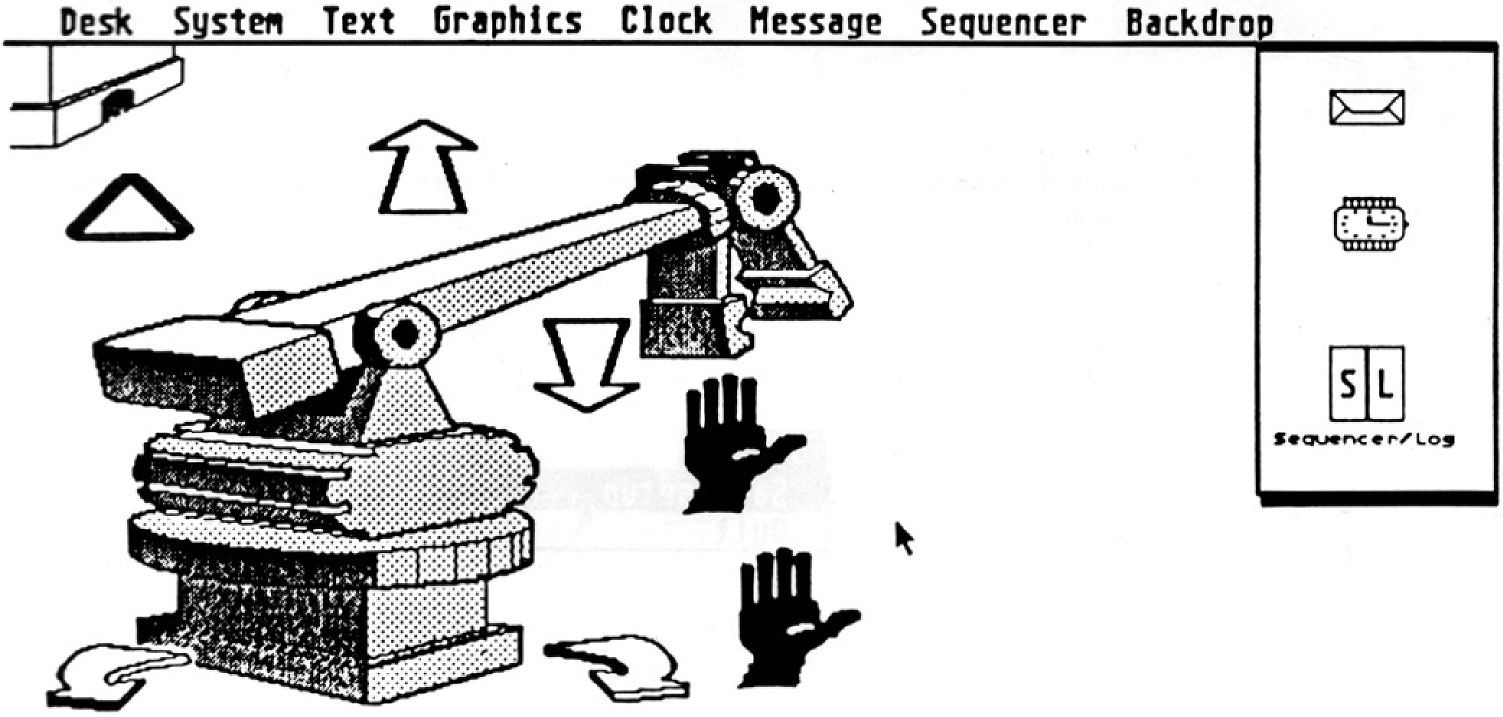
command Mt
motor#
speed
dir
time
reply_to

menu
Cancel
Accept

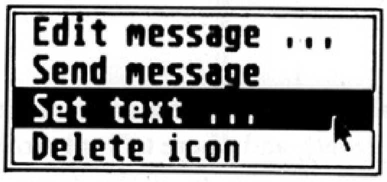
The Message Edit Box this time shows a different message to be edited. The grey text is the same, and you still need to fill in the number of the motor you're sending the message to, but you also have to specify speed (a number between 1 and 4, 4 being the fastest speed), direction (either -1 or 1), time (a number of tenths of a second - try starting with 10 so that you get one second's worth of movement), and where to 'reply-to' (ignore this for the moment, and leave that line blank). When you've done all this, click on 'Accept', and you'll be back onto the backdrop of the robot - ready to start some movement. Now click on the Arm-Up icon, and the message you just edited to be associated with that icon will be sent to a motor, causing it to move for a certain amount of time. You may find that the wrong motor moves, or that it moves in the wrong direction or for the wrong time; double click on the icon, choose 'Edit Message' from the pop-up menu, and you'll see the Message Edit Box again, this time with the message data in place, just as you entered it; the system has filled in the 'reply_to' line with 'no-slot'; carry on ignoring this. Edit the rest of the message as you like, and 'Accept' it.

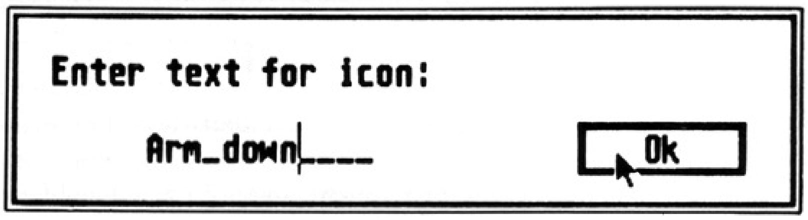
Once you've got one icon doing the right thing, it should be easy to attach appropriate messages to the others, so that you can drive the Mini_Arm around using only the mouse. If you haven't already done so, add appropriate icons to move the base motor - pehaps using the curved arrows to indicate clockwise and anticlokwise rotation.

It is also a good idea to add icons to stop the arm moving - the 'motors' message is 'motor_stop', and you'll need a separate icon and stop message for each motor. You may even want to add an icon for the initialisation messages we sent earlier, so that you'll remember to initialise next time you use ROBOKIT. Your screen might now look similar to this:



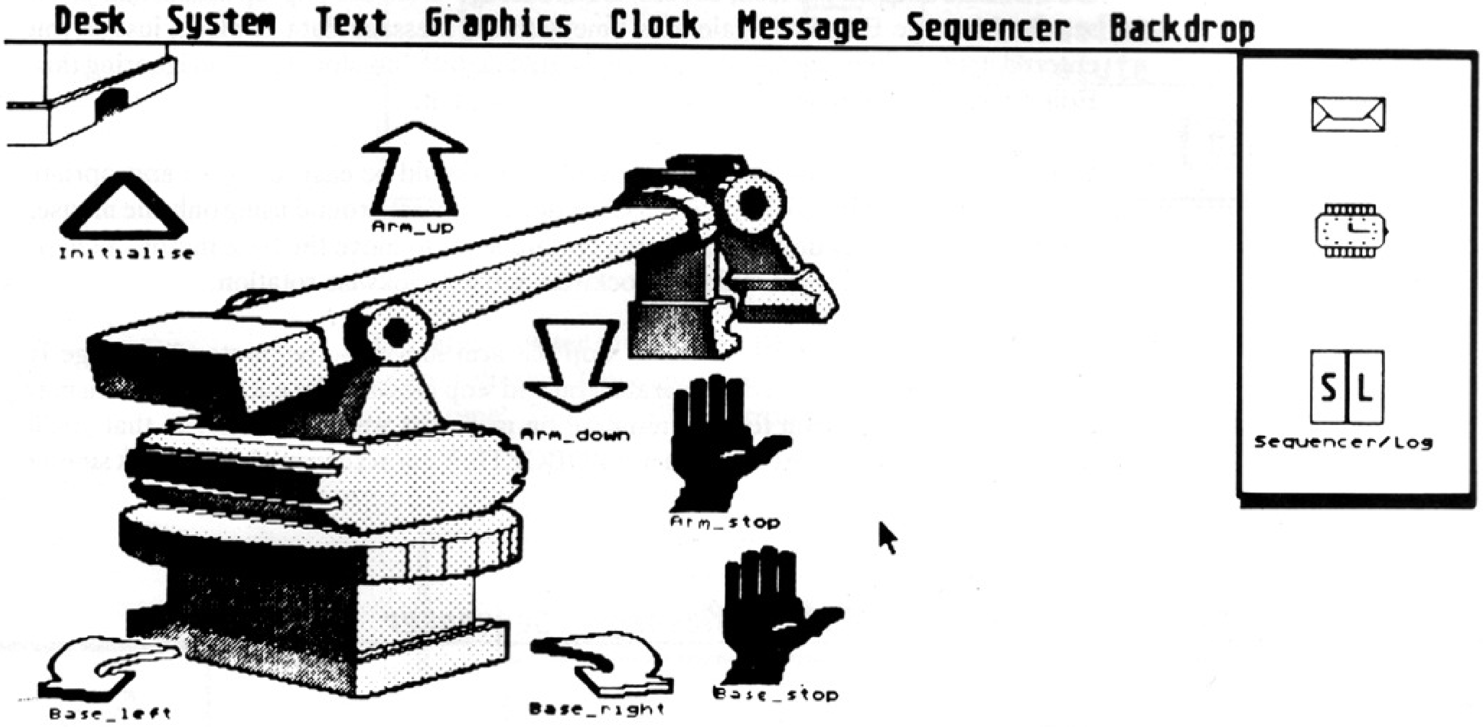
With so many icons on the screen you may want to label them, in which case you double-click on one and choose 'Set Text', then type the label for the icon in the edit-box that appears.


Edit message ...
Send message
Set text ...
Delete icon

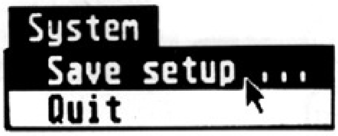

Enter text for icon:

Arm_down____

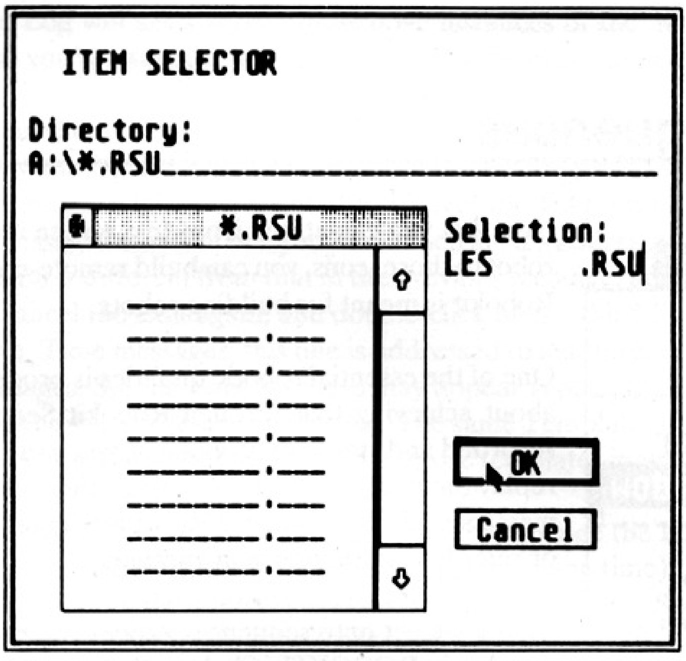
This gives a much more pleasing result, perhaps like this:


Desk
System
Text
Graphics
Clock
Message
Sequencer
Backdrop
Initialise
Arm_up
Arm_down
Arm_stop
Base_left
Base_right
Base_stop
S
L
Sequencer/Log

Lastly, you should save your system, by choosing 'Save setup' from the 'System' screen menu.


System
Save setup ...
Quit

This enables you to create a .RSU file containing your choice of backdrop images, your icons, and their positions and messages. Next time you run Robokit you can select this .RSU file at the start of the session, and thus regain the Backdrop images, Icons and Messages that you created in this session. So type in an appropriate name if you want to save your work or cancel if you don't.



ITEM SELECTOR

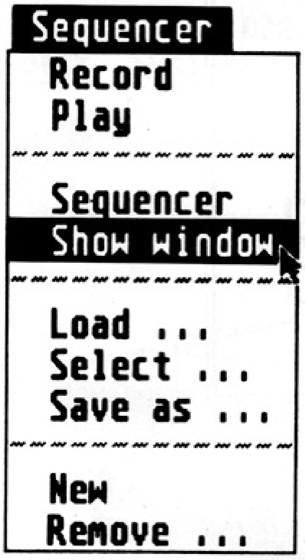
Directory:
A:\*.RSU

Selection:
LES .RSU

.RSU files do not contain Backdrop and Icon images -only references to them; Robokit will look on the current disk drive for .DRP files containing these images. If the disk does not contain these files you will see an Error Box.

OK the Error, then either Quit Robokit and rearrange your disk files, or carry on in Robokit, loading your Backdrops afresh, adding Icons, attaching Messages, and so on as before. In the reference section of this manual, there are some further notes concerning file types and locations for those who wish to know.

3.2. Sequences

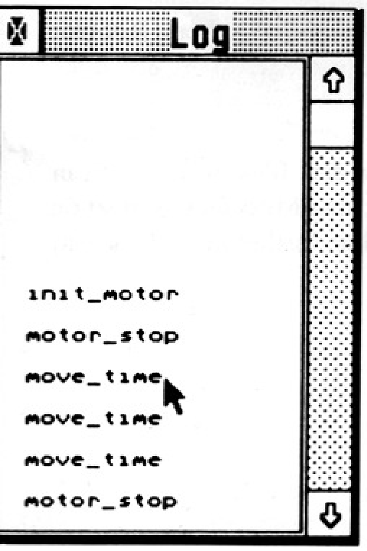


Now that you can attach icons to a screen image, and attach control messages for your robot to those icons, you can build remote-controlled automata - but that's just the start; Robokit is meant for building robots.

One of the essential robotic qualities is programmable repetitive action. This chapter is about achieving that through Robokit Sequences. These are sequences of messages, recorded and stored by Robokit as you issue them; you can name these Sequences, and replay them, causing the robot to go through the sequence of actions that your original messages initiated and with the same timing. You can attach Sequence names to icons, or refer to one Sequence in another.

Before we get onto sequences proper, we need to understand the function of the message log in ROBOKIT. Click on the small box marked 'L' in the control panel.

A small window entitled 'Log' appears on the screen. (you could have achieved the same thing at this point by pulling down the Sequencer menu and clicking on Show Window).



```
init_motor
motor_stop
move_time
move_time
move_time
motor_stop
```

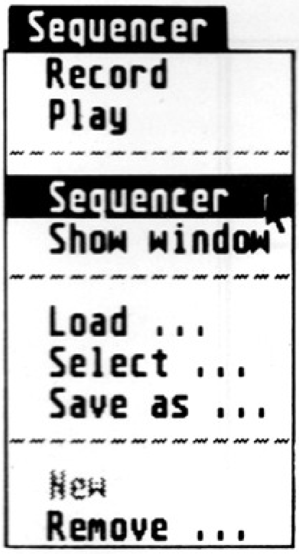
You can drag the Log, and close it, like any other Gem window; you can't re-size it, however. The Log itself is what its name implies - a log (or diary) of the Robokit messages that you issue.

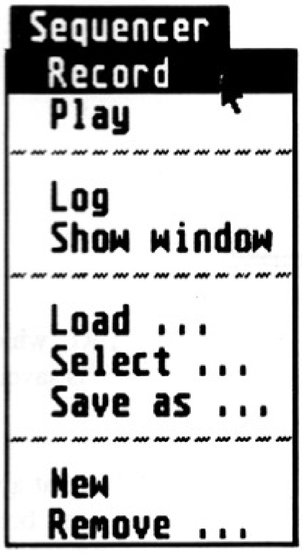
Try sending some messages, either by clicking on a live icon, or by using the envelope for sending messages directly. You'll see that message names appear at the bottom of the Log as you send them, and the Log contents scroll upwards, eventually disappearing off the top of the window. The Log, therefore, always shows a sequential record of the 10 most recent messages sent, with the most recent at the bottom. This is useful enough in itself, but there's more to the Log than that.

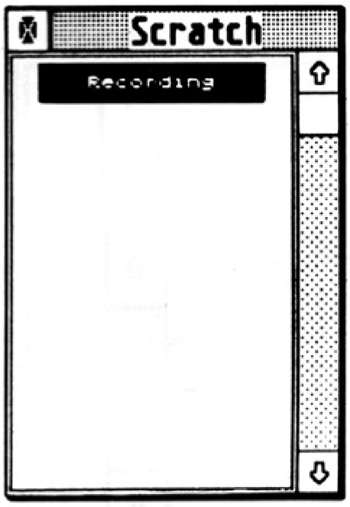
Choose a message in the Log, one that has some definite and observable effects on your system - a move-time message, for example, from the Up-Arm icon. Click on that message in the Log. The message is sent to the Mini_Arm, as if from an icon or the Send option; the Log window doesn't change, however. That rather spoils the record-keeping aspect of the Log, but it means that if there's a a useful message or group of messages in the Log, you can repeatedly send them just by clicking in the Log, and they won't scroll off the window. This can be a great convenience when you're trying to get the data in a message just right, or when you're adjusting some mechanical aspect of the system. A message in the Log becomes, in effect, an instant text icon.

If you double-click on a Log entry you get the Edit Message Box; it contains the details of the message that you clicked on. You can edit this message, and send it by clicking Accept. This causes a new instance of this message name to appear at the bottom of the Log - unlike single-clicking a Log message, which sends the message but doesn't change the Log.

Log entries may have puzzled you: different icons can cause the same message name to appear in the Log. If you click on an Up-Arm icon then a Down-Arm icon then a Left-







Arm icon, the Log will show three consecutive instances of the 'Move-Time' message (assuming that you're using the set-up suggested in the previous section).

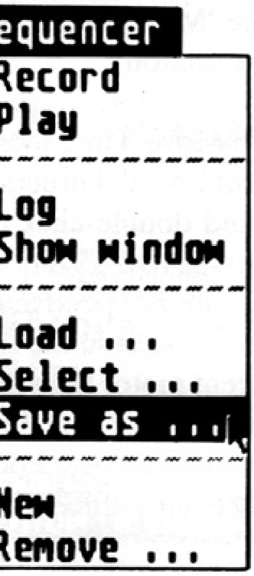
Double-click on the earliest (highest in the Log) of those three Move-Time messages; it's addressed to the Up-Down motor, and has whatever values for Speed, Direction and Time that you filled in when you edited it. Cancel the Edit, and double-click on the Move_Time message below the one you just looked at - it's addressed to the same motor, but the Direction is different from that in the previous message, and the speed and time may be, too. Cancel the Edit again, and double-click on the third (lowest in the Log) of the three Move_Time messages; this one is addressed to a different motor than the previous two messages. So, the same message may appear repeatedly in the Log, but may not actually be the same message every time. The same Template or Message Form has been used every time, but the data entered in those templates may have differed.

To start recording and playing sequences, we need to change the Log window to a Sequencer window (you can't have them both open at the same time). You can do this by clicking on the 'S' box in the control panel, or by pulling down the Sequencer menu and clicking on Sequencer. Either way you'll see the Log window re-named as Scratch - because we haven't specified any other name for the sequence yet. Issue a few more messages, as you did when trying out the Log window, and you'll see that this Sequencer window doesn't behave like the Log - nothing appears in it. Switch the window back to its Log function (by clicking on 'L' in the control panel, or pulling down the Sequencer menu and choosing the Log) and you'll see that, though invisible, the Log has been continuing to record your messages. Return to the Sequencer Scratch window.

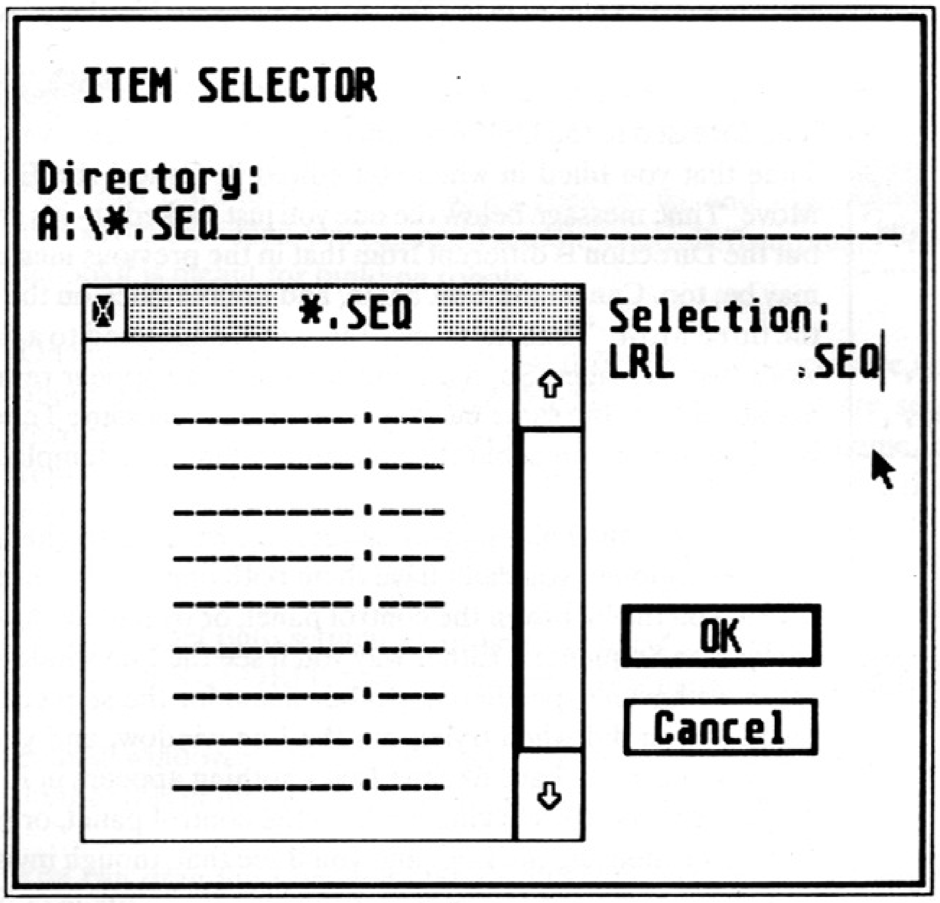
Now pull down the Sequencer Menu and click on Record.

The Scratch window now shows a black bar saying 'Recording'. In this state it will behave like the Log, and record every message sent. Using the icons, send a few messages - say Left-Arm, Right-Arm, Left-Arm. Remember that the sequencer records messages in real time : if you pause between clicks, these pauses will be recorded also. Recording starts as soon as the recording bar shows up in the sequencer window. Pull down the Sequencer menu - the Record option has a tick against it, echoing the Recording marker in the Scratch window.

Click on Record, and the Recording marker disappears from the Scratch window. The contents of the window are a Sequence, a series of Robokit messages. Now we have the programmable repetitive action referred to at the start of this Chapter - a Sequence is a program that can be executed at any time, and that will always behave in the same way. Creating Sequences is programming your robot.


equencer
Record
Play
Log
Show window
Load ...
Select ...
Save as ...
New
Remove ...

Pull down the Sequencer menu and select Save as.

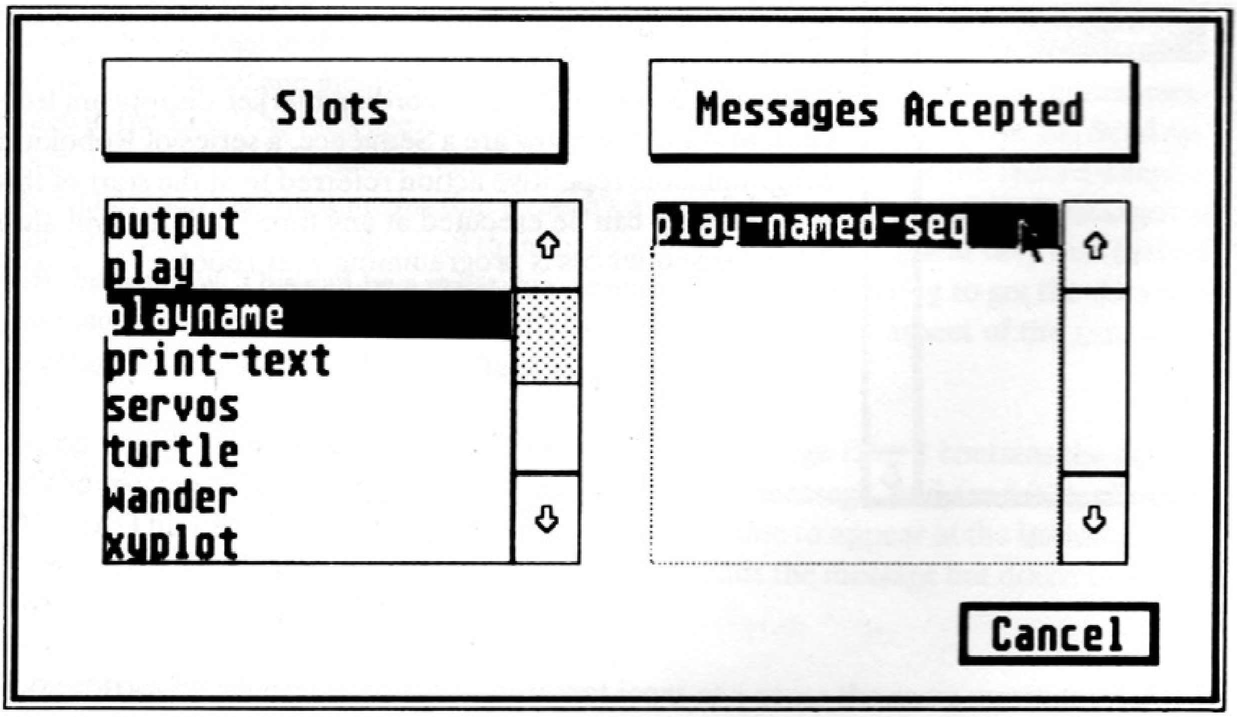

ITEM SELECTOR
*.SEQ
OK
Cancel

Directory:
A:\*.SEQ

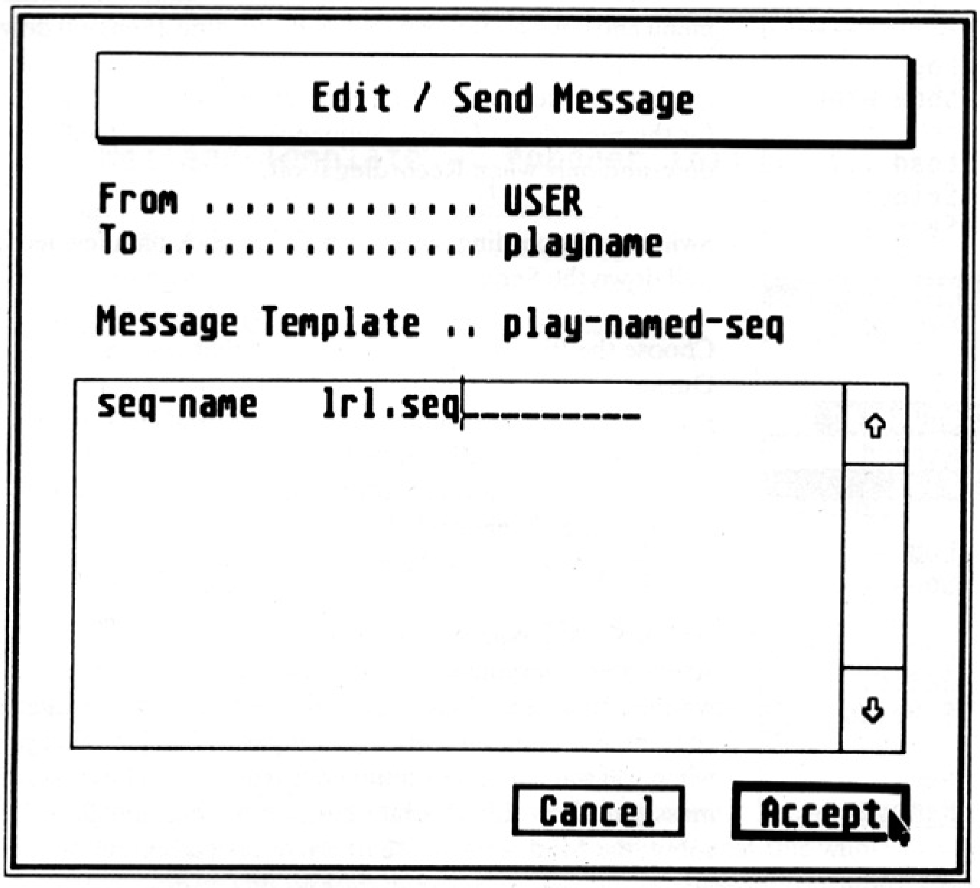
Selection:
LRL .SEQ

A typical Gem file selector appears. Type in a name for your Sequence - say LRL, which Robokit automatically amends to LRL.SEQ - and click on OK. The Sequence is saved to disk, and the Scratch window is renamed LRL.

Now add a new icon to the Backdrop, and in the Message Destination window that appears after you've selected an icon, choose Playname. In the Messages Accepted box choose Play-Named-Seq.


Slots
output
play
playname
print-text
servos
turtle
wander
xyplot
Messages Accepted
play-named-seq
Cancel

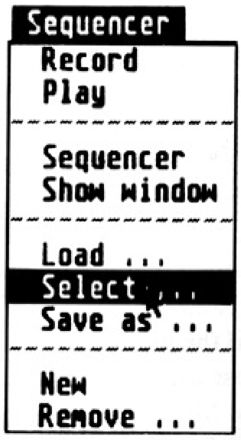
Finally, in the Edit Message Box type LRL as the Seq_ Name. Now when you click on the new icon, all the messages in the LRL Sequence will be issued.



From .... USER
To .... playname

Message Template .. play-named-seq

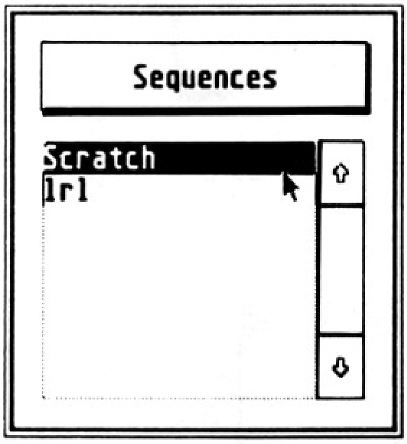
seq-name   lrl.seq



Before you test your new icon you should create an icon that will halt execution of a sequence, since there is no quick way of doing this from the screen menus. Add a suitable icon to the Backdrop, select Play as its destination slot, and Stop_Play as the message. This will do just what it says - stop a sequence from playing, and return control to you.

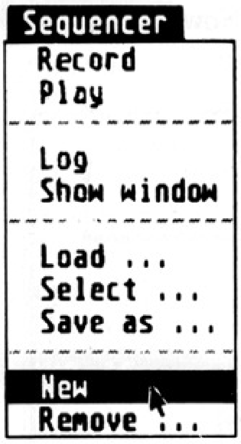
If you have to stop a sequence in an emergency, and don't have a Stop_Play icon on the Backdrop, you can pull down the Clock menu and Halt the Clock; unfortunately, if Robokit is executing a Clock-dependent message (such as Move_Time, Move_Click or Move_Dist) when the Clock is Halted, then the motors affected will not be stopped, but will go on turning for ever. A Stop_Play message will interrupt a Sequence, but won't interrupt an individual action - a motor that's been sent a Move_Time message, for example, will run until that time is expired.

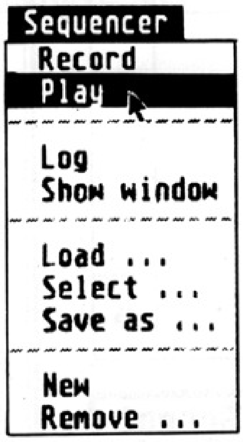
For complete safety, therefore, your Emergency Stop icon should play a Sequence consisting of Motor_Stop messages for all motors, and a Stop_Play message. That should stop just about anything.



Continuing your investigation of Sequences, pull down the Sequencer menu, and choose the Select option.

A selector box entitled 'Sequences' appears containing the names of the Sequences currently available. Choose Scratch. The LRL window is replaced by a blank Scratch window, ready to record a new Sequence.


Sequencer
Record
Play
Log
Show window
Load ...
Select ...
Save as ...
New
Remove ...


Sequencer
Record
Play
Log
Show window
Load ...
Select ...
Save as ...
New
Remove ...

Set Recording on, and click on a few icons. Message names scroll up the Scratch window as before; if you click on the LRL icon, the Play-Named-Seq message appears in Scratch. When you've collected a few messages in Scratch, pull down the Sequencer menu and click on Record to stop Recording, then pull down the Sequencer menu again.

Choose the New option: the contents of the Scratch window disappear, leaving it blank for the recording of a new Sequence. This erase facility works only in the Scratch window, and only when Recording is off.

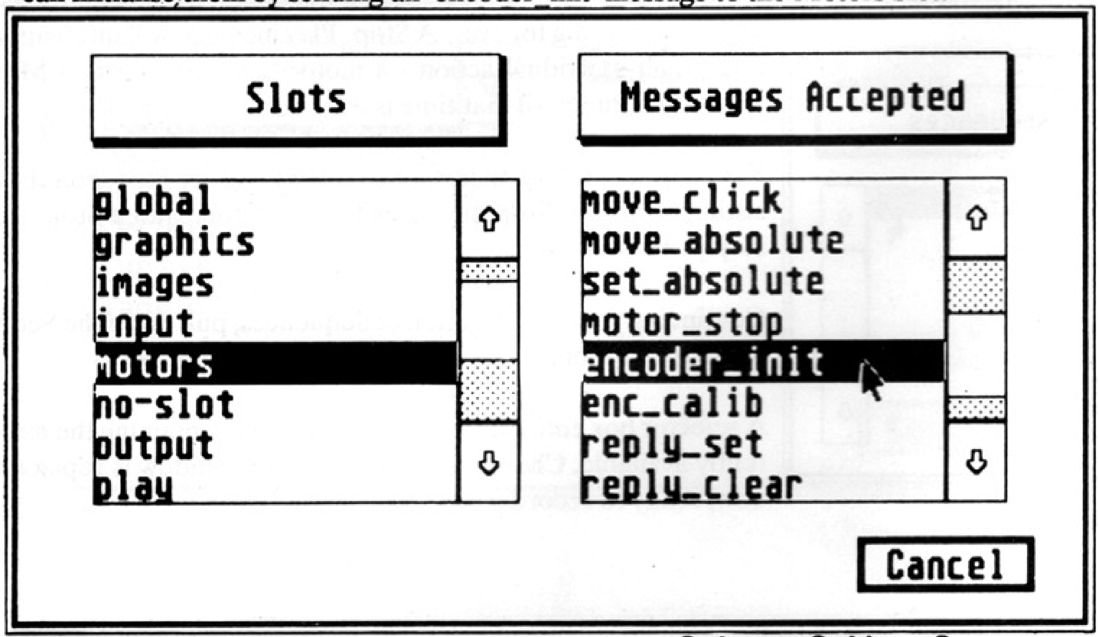
Switch on Recording, and - as before - click on a few icons. Switch off Recording, and pull down the Sequencer menu.

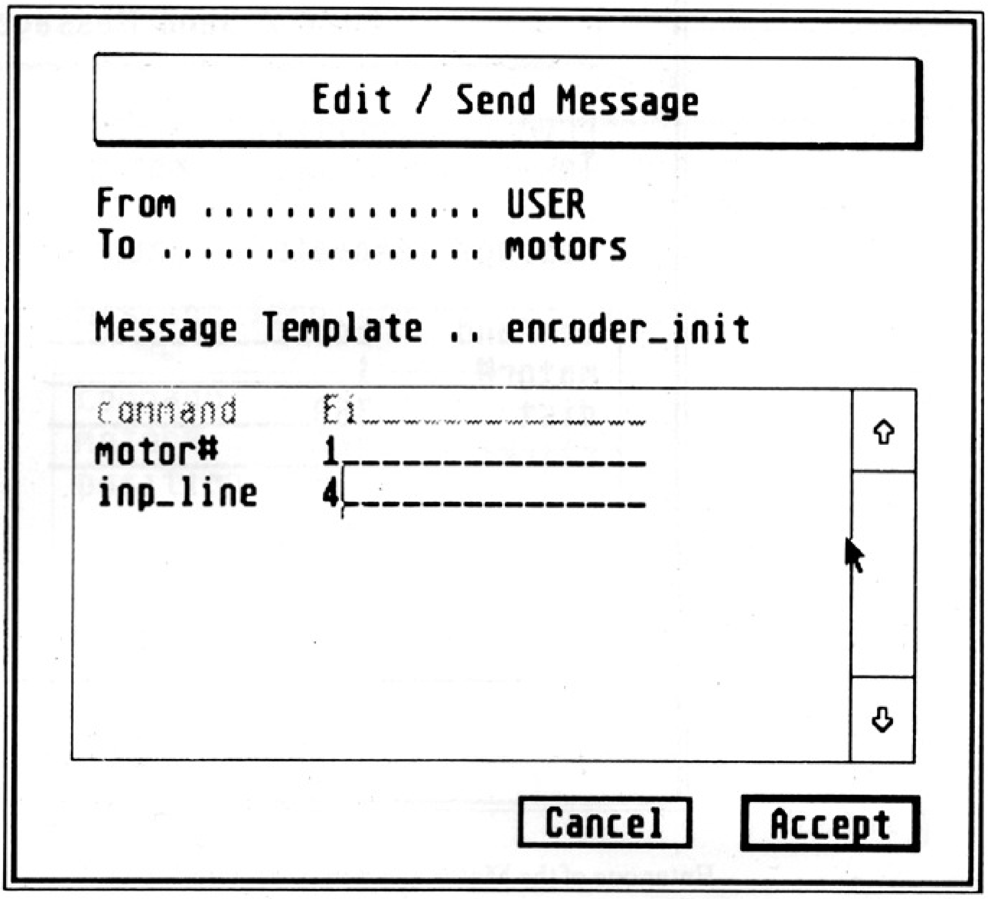
Choose the Play option: the Sequence that you've just recorded in Scratch is executed. During this Sequence a Playing marker appears in the window, and disappears when execution ends. Switch on Recording again, and click on an icon: the new message appears at the bottom of the Scratch Sequence, showing that it has been tacked onto the end of the Sequence already recorded. If you switch off Recording, and switch Playing on, the new longer Sequence will be executed. In this way you can build up Sequences step by step, pausing to check the results of each step.

An important aspect of Sequences as we've created them so far has probably already struck you - everything is relative, in that execution of any message or Sequence proceeds with the Mini_Arm in whatever position the last message left it. Sometimes this is unimportant, sometimes it's vital, but sometimes it's just what you don't want. There are times when you want action to commence from some known starting position. Several Robokit messages allow this absolute positioning, but they do rely on position reports from the robot; the Mini-Arm has shaft-encoders on both motors for this purpose.

Before you can use the encoders, you have to initialise them, a process just like initialising the motors, described in Chapter 1. A shaft encoder is simply a device that measures the rotation of a shaft in terms of units or 'clicks' as we call them. You can make an encoder from a bent paper clip (as we show in Chapter 2) or you can use the proper Lego shaft encoders that work by bouncing infra-red off a black-and-white marked disc.

Assuming that you've built the mini-arm project and connected the two encoders to two of the input lines (if you are using the Lego encoders you must use lines 0 and 1), you can initialise them by sending an 'encoder_init' message to the Motors Slot.


Slots
Messages Accepted
global
graphics
images
input
motors
no-slot
output
play
move_click
move_absolute
set_absolute
motor_stop
encoder_init
enc_calib
reply_set
reply_clear
Cancel


Edit / Send Message
Cancel
Accept

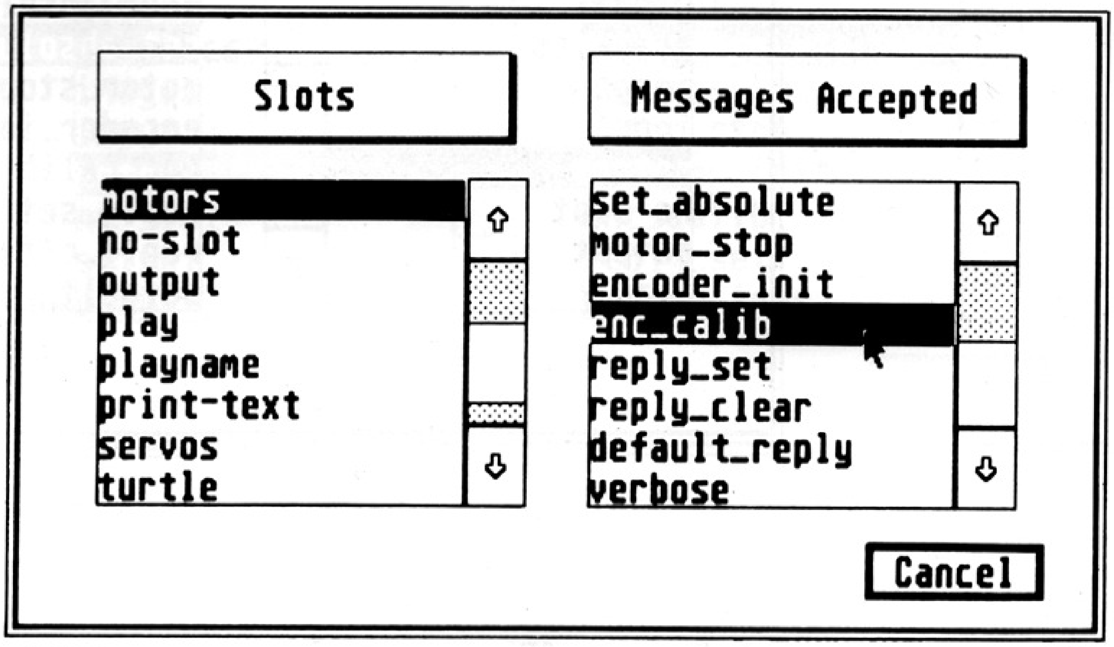
From .... USER
To .... motors

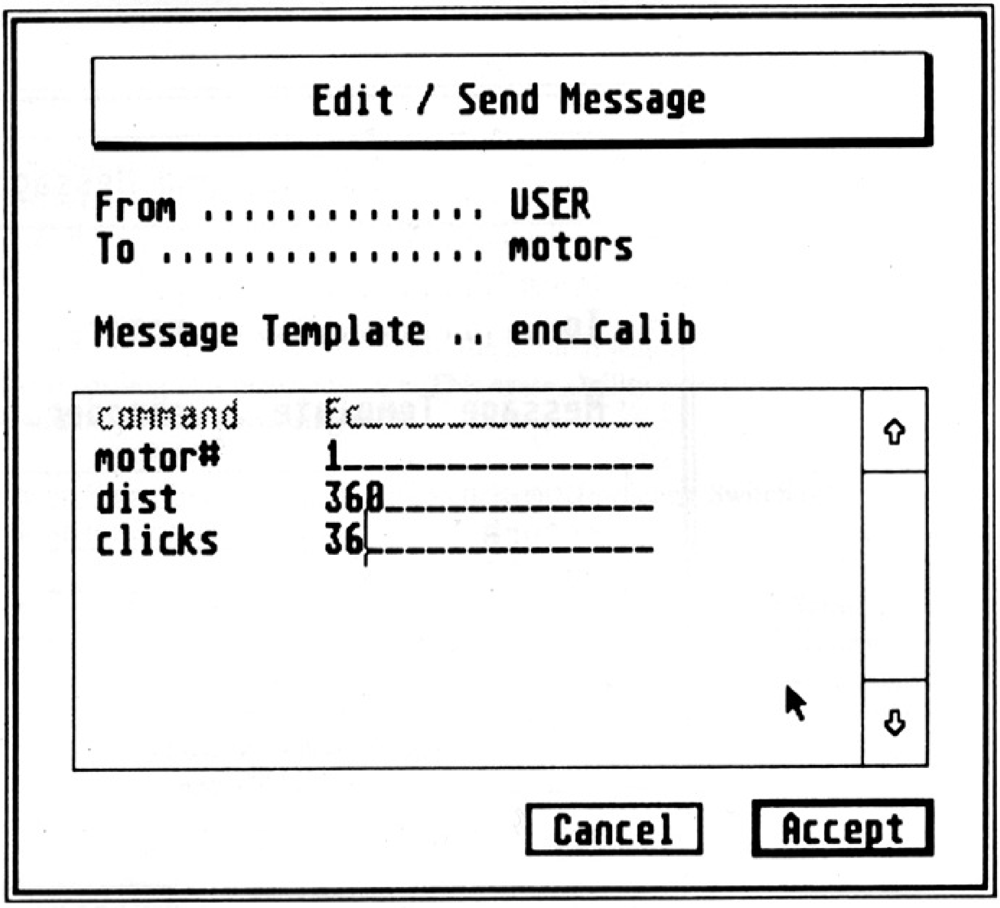
Message Template .. encoder_init

command Ei
motor# 1
inp_line 4

In the Edit Message box, type the motor number, and the input line number for the encoder that corresponds to that same axis of movement. Repeat this whole process for each motor/encoder pair on your model. For example, the base motor (motor 0) may be connected to outputs 0 and 1, with the shaft encoder connected to input 0. The arm motor (motor 1) may be connected to outputs 2 and 3, with its shaft encoder attached to input line 1.

When you've initialised the encoders, you've got to calibrate them, which means attaching some unit of measurement to the clicks of the encoder micro- switches. Put the Mini_Arm in some recognisable position, such as pointing at you with the Arm pointing up as high as it will go. Send an Enc-Calib message to Motors.


Slots
motors
no-slot
output
play
playname
print-text
servos
turtle
Messages Accepted
set_absolute
motor_stop
encoder_init
enc_calib
reply_set
reply_clear
default_reply
verbose
Cancel


Edit / Send Message

From .... USER
To .... motors

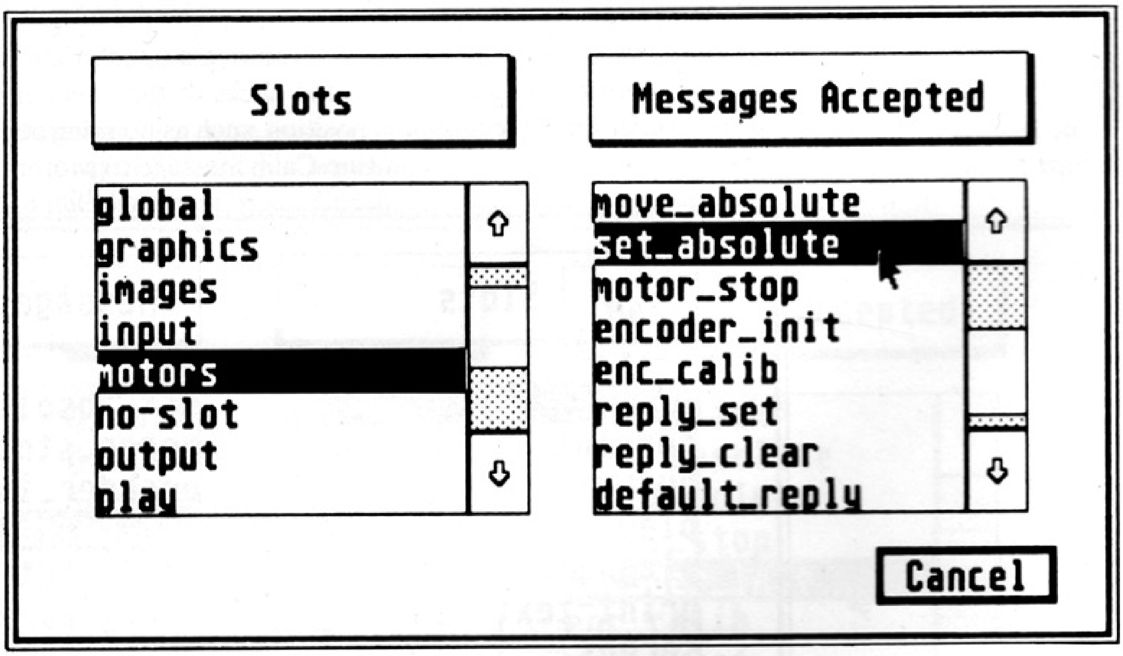
Message Template .. enc_calib

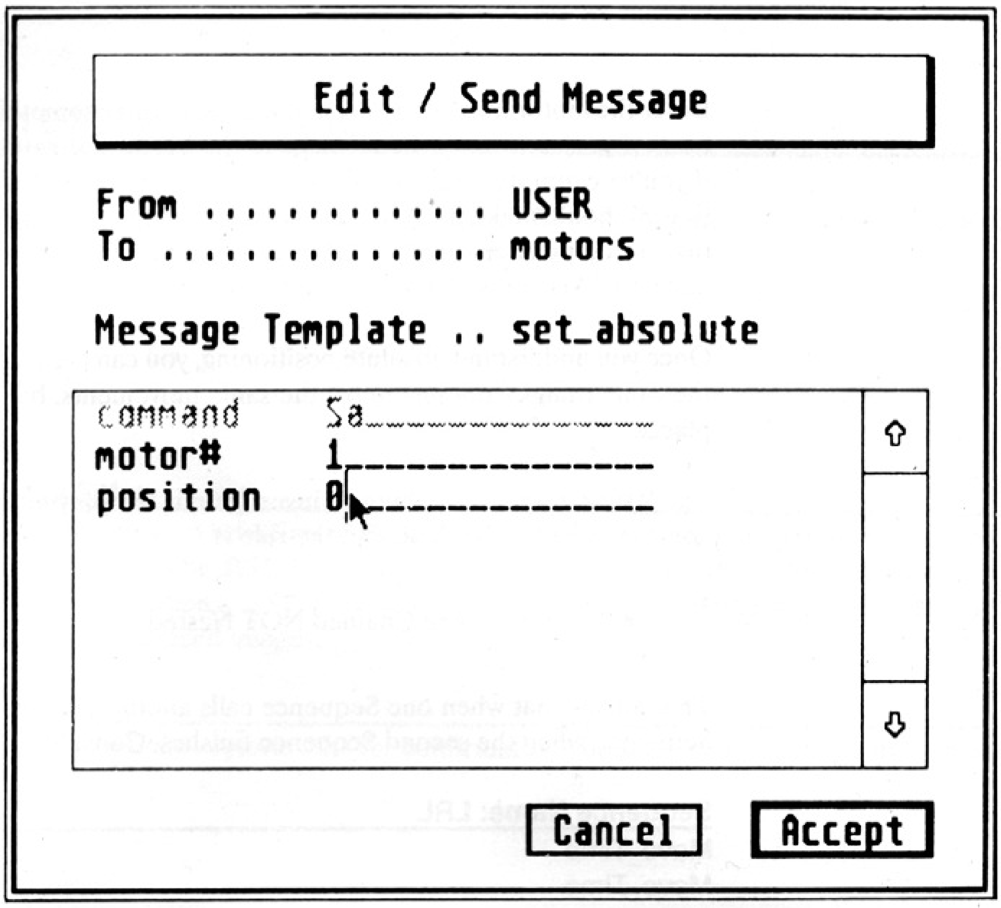
command Ec
motor# 1
dist 360
clicks 36

Cancel
Accept

Enter one of the Motor numbers as required. Now decide what distance you want a click of that encoder to represent - When the Mini_Arm turns a complete circle about its vertical axis, for example, the encoder clicks 36 times; you might chose, therefore, to say that 36 clicks represent a 'distance' of 360 (the number of degrees in a turn).

Choose any distance and number of clicks that make sense to you, and Accept the message. Do the same thing for the other motor, again choosing whatever you like for distance and clicks. Now send the Set_Absolute message to Motors.


Slots
global
graphics
images
input
motors
no-slot
output
play
Messages Accepted
move_absolute
set_absolute
motor_stop
encoder_init
enc_calib
reply_set
reply_clear
default_reply
Cancel



Edit / Send Message

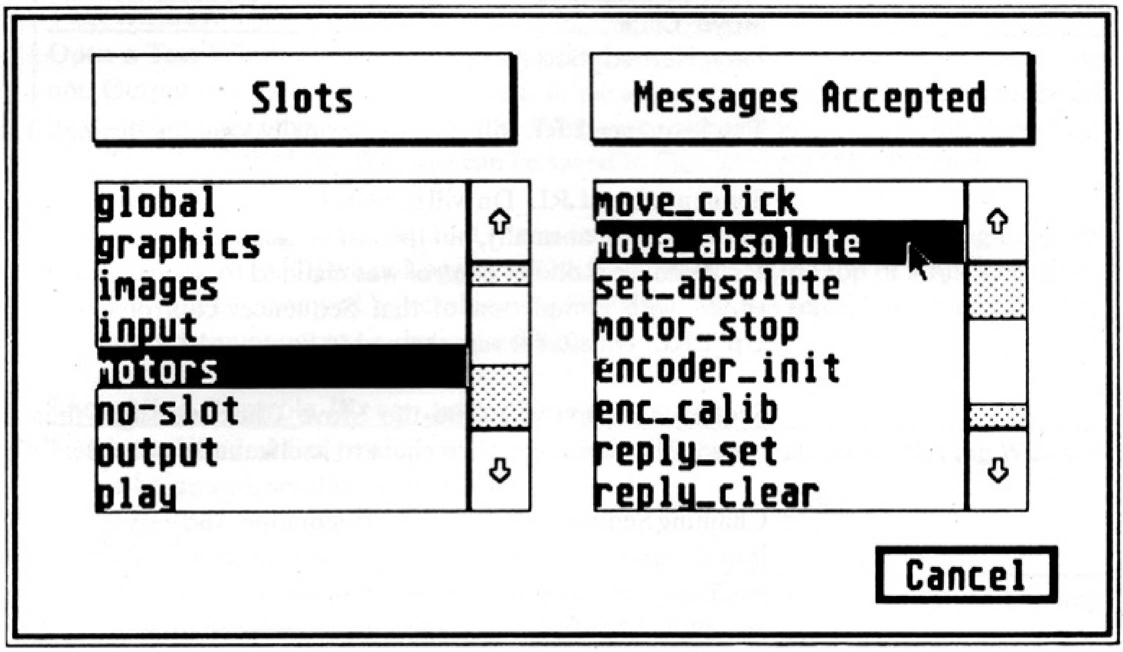
From .... USER
To .... motors

Message Template .. set_absolute

command Sa
motor# 1
position 0

Cancel
Accept

Enter the motor number, and make the position number zero. Accept the message, and do the same for the other motor. The current position of the Mini_Arm is now Position Zero for both motors, and all absolute movements are made with respect to these zero positions. Send a Move_Absolute message to Motors.



Slots
Messages Accepted
global
graphics
images
input
motors
no-slot
output
play
move_click
move_absolute
set_absolute
motor_stop
encoder_init
enc_calib
reply_set
reply_clear
Cancel

Enter the motor number and speed as usual. Enter something recognisable for the position - if you're sending the message to the horizontal-rotation motor, for example, and if you've calibrated it to 36 clicks equals 360, as suggested, then Move_Absolute Position 90 should make the Arm turn horizontally through a right-angle from the Zero position in the positive direction. If you now send Move_Absolute Position 0, this should make the Arm move back to the Zero position.

Once you understand absolute positioning, you can create sequences that will always do the same things - not just make the same movements, but move to and from the same places.

The last Sequencer attribute to investigate is the playing of one Sequence from inside another. This can be done, but the rule is

- Sequences Are Chained NOT Nested

That means that when one Sequence calls another, control does not return to the first Sequence when the second Sequence finishes. Consider these examples:

Sequence Name: LRL

Move_Time
Move_Time
Move_Time

Sequence Name: Up_LRL_Dn

Move_Click
Play_Named_Seq (LRL)
Move_Time

Sequence Name: Loopy

Move_Click
Play_Named_Seq (Loopy)

The Sequence LRL will execute normally, running through its three messages.

Sequence Up_LRL_Dn will send its first message, Move_Click, then Sequence LRL will be performed normally, but the last message in Up_LRL_Dn, Move_Time, will not be sent because Robokit control was chained to Sequence LRL, and Sequence execution ended with completion of that Sequence: control never 'came back' to Sequence Up_LRL_Dn after it was chained to Sequence LRL.

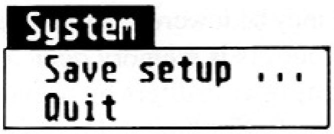
Sequence Loopy will send the Move_Click message, then chain to itself, send the Move_Click message, then chain to itself....and so on ad infinitum.

Chaining Sequences is a powerful technique, and easy to use once the simple Chain Rule is understood. If you have problems with Sequences, then open the System Window from the Text menu, and send the Verbose message to Motors, and to any other slots that will accept it. Then, when you run a Sequence, the messages in the System window will tell you what Robokit thinks it's doing at any time, a useful cross- check on what you see your robot doing. Or not.

3.3. Menus - reference guide

This section explains the functions offered from the menus at the top of your Robokit screen.

System Menu



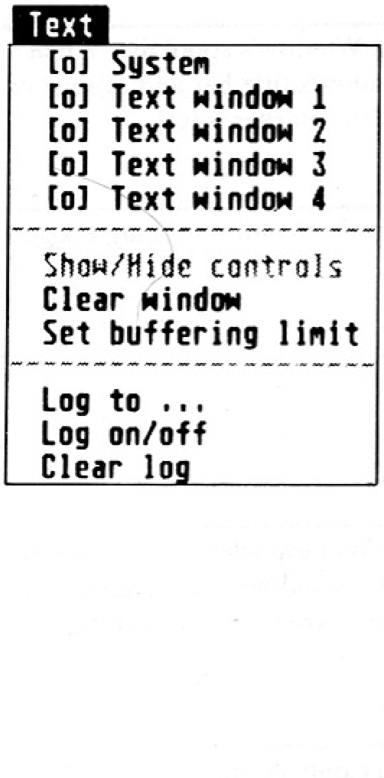
Save Setup

Save the current selection of Backdrops, Icons, Messages and Windows in a .RSU file for later use. The .RSU file contains references to Backdrops and Icons, not the images themselves; when a .RSU file is loaded at Run-time, Robokit looks for the files containing the referenced images.

Quit

Return to GEM. System asks for confirmation but does not remind you to Save Setup.

Text Menu



System

Open the System Text Window. Displays the System Window in its current position and state. This Window is a Text Window like the others but, being designated for System use, is more likely to carry diagnostics and status reports than the other Text Windows. Can be Logged, like the other Text Windows.

Text Window 1-4

Open a Text Window. Slots can output text to any Text Window whether it is open or not. Output to a Text Window proceeds in parallel with all other tasks being executed by Robokit. Users can output to a Text Window by sending messages to the Print-Text slot. The contents of any Window can be saved to file - see Log On/Off below.

Windows are flagged [*] in the Text Menu and on their Title Bars when being Logged; the flag changes to [O] when Logging is Off. Text scrolls off the top of a full Text Window, but can be reviewed by scrolling the Window with the scroll bars. Output to the Window continues correctly during scrolling.

Show/Hide Controls (Reserved for future versions)

Toggle the Window Controls in the active Window. When Controls are Off the Window cannot be moved, scrolled or re-sized.

Clear Window

Clear the active Window, copying any Text to the Log file, if on. Text erased in this way cannot be recaptured, except from the Log file.

Set Buffering Limit

Set the number of lines of text in the Text Windows' memory buffers. Text output to Window is stored in an area of RAM dedicated for that Window's use, called a memor buffer. Text is stored in the buffer in the order in which it appears until the buffer is ful thereafter, incoming text causes the oldest text to scroll out of the buffer; if the Windo is being Logged, the scrolled Text is copied to the Log file, otherwise it is simply erase

The scroll bar on the Text Window moves a pointer up and down the buffer, and t Window displays the contents of the buffer from that pointer onwards.

The maximum Buffer size is 100 lines per Buffer; this limit may be lowered automatica ly at any time by Robokit, since memory reserved for the buffers is memory denied t Robokit. If Robokit needs memory it will start reducing any open Buffers by scrollin Text out of them (thereby, in effect, temporarily lowering the Buffer limits). If reducin the Buffers doesn't produce enough free memory, Robokit will print OUT O MEMORY in the active Text window every time you try to write to that Window. Thi should not stop Robokit from performing normally in other ways.

Log To

Open a spool file on disk to which all text output to the current Text Window will b saved. The file so created is a simple ASCII file which can be VIEWED in GEM an edited in any word processor or text editor..

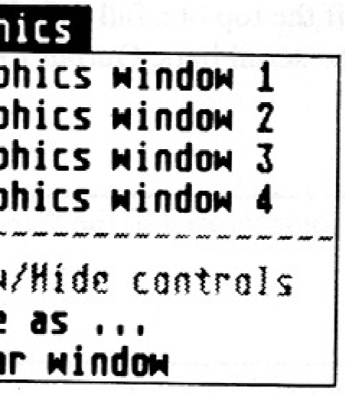
Log On/Off

Toggle the writing of Window output to the current Text Window's spool file. When Window is Logged On a marker, [*], appears on the Window's title bar and against it name in the Text menu; when the Window is Logged Off the marker is [O].

Clear Log

Delete the Log file for the current Window - provided that the Window is being Logge at the time, of course.

Graphics Menu



Graphics Window 1-4

Open a Graphics Window. Slots can output to a Graphics Window whether it is open or not, and this output proceeds in parallel with all other tasks. Windows can be moved, re-sized, opened or closed without affecting output. Users can output graphics to Graphics Windows by sending messages to the Graphics slot.

Show/Hide Controls (reserved for future versions)

Toggle the Window Controls in the active Window. When Controls are Off the Window cannot be moved, scrolled or re-sized.

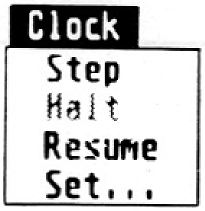
Save As

Save the current Graphics Window to disk as a .IMG file. In this format the contents of Graphics Windows can be displayed and edited in GEM Paint, and viewed and printed through GEM Desktop Output. Other art packages may accept .IMG files or allow their conversion to acceptable formats.

Clear Window

Clear the current Graphics Window. Output erased in this way cannot be regained.

Clock Menu



Step

Make the (Halted) System Clock generate one tick. Allows step-by-step execution of all Robokit processes. Can be useful when debugging Sequences or hardware, but really designed for programmers who write their own Robokit slots.

Halt

Stop the System Clock. Halts all Robokit processes until the Step or Resume options on this menu are chosen. System Clock is Halted by default when Robokit is run; no processes will execute, therefore, until Clock is Resumed or Stepped.

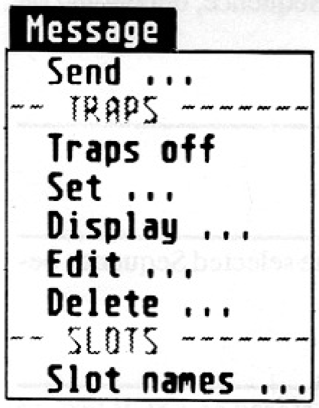
Resume

Re-start the (Halted) System Clock. The Clock Resumes operation at the speed to which it was last Set. When Robokit is Run, the Clock is Halted by default; no processes will execute, therefore, until the Clock is Resumed or Stepped.

Set

Set the speed of the System Clock. Robokit's default Clock speed is 100 ticks per second; this may be changed to 50 tps, 25 tps, 10 tps, or 0 tps. Execution of all processes will be correspondingly affected.

Message Menu



Send

Send a message to a Robokit slot. All Robokit activity depends on the transfer of messages; this is how slots input and output data and reports; this is how users control the system.

Current slots are listed in the Selector Box that appears when the Send option is chosen. When a Destination slot is chosen, the messages which it will accept are listed in a second Selector Box.

Traps Off

Turn off all message traps. Traps, whether on or off, can be Set, Displayed, Edited or Deleted, as described below.

Set

Set a trap for any kind of message from any slot to any slot.

Display

Display the list of current traps.

Edit

Edit any trap selected from the current list.

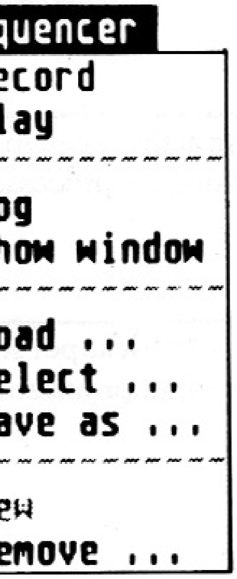
Delete

Delete any trap selected from the current list.

Slot Names

Display the names of all current Robokit slots.

quencer Menu



Record

Toggle recording of messages in the current Sequence Window. Recording proceeds from the end of any Sequence currently listed in the Window. A Recording marker is displayed in the current Window and on the Sequencer menu.

Play

Execute the Sequence in the current Sequence Window. A Playing marker is displayed in the current Sequence Window. Playing can be stopped by Sending the Stop_Play message to the Play slot. This will not interrupt execution of a message sent by the Sequence, however.

Sequencer or Log

Toggle Log and currently selected Sequence Window. Toggling occurs whether the Window is shown or not.

Show Window

Display the Sequence Window on-screen - either the Log or a Sequence, depending on the Sequencer/Log toggle.

Load

Load a named Sequence from disk.

Select

Select a named Sequence from the list of current Sequences. The selected Sequence becomes the contents of the current Sequence Window.

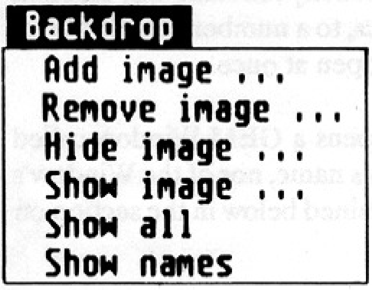
Save As

Save the current Sequence Window to disk. Sequences are not stored as ASCII text, so cannot be Viewed on the Gem Desktop - use a debugger or file dump utility. Now Clear the Sequence Window. Effective only in the Scratch Window, and only when the Window contents have not been saved.

Remove

Remove a named sequence from the list of current Sequences.

Backdrop Menu



Add Image

Load a named .IMG file from disk and display it on the Backdrop. A maximum of 16 images can be loaded and displayed. Images can be dragged around the Backdrop with the mouse. Double-clicking on an image allows an icon to be attached to the image.

Remove Image

Remove a named image (and any attached icons) from the Backdrop. Once Removed, an image cannot be displayed again until reloaded from disk. Its icons can be saved only by the Save Setup option on the System menu.

Hide Image

Hide a named image (and its icons) from view on the Backdrop; Hidden images can be Shown again without their being re-loaded from disk. Show Image Display a named Hidden image on the Backdrop.

Show All

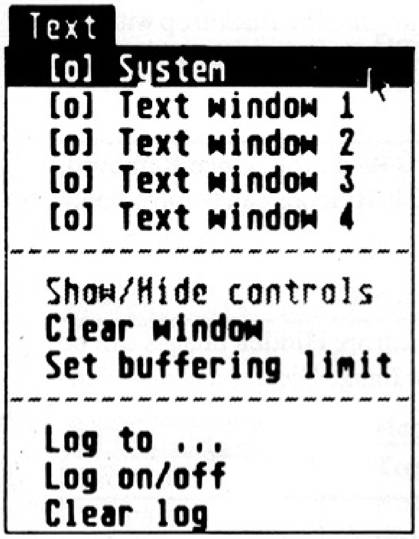
Display all Hidden images on the Backdrop.

Show Names

List the names of all current images, whether Hidden or not.

3.4. Text

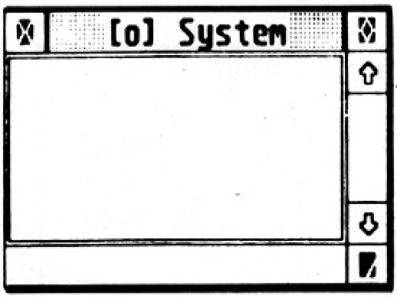
We have seen in previous Chapters how Robokit enables you to send messages to Slots which control physical devices plugged into your Atari; in exactly the same way Robokit enables you to send output messages, both text and graphics, to a number of GEM Windows on the Backdrop. Up to nine such Windows can be open at once.


Text
[o] System
[o] Text window 1
[o] Text window 2
[o] Text window 3
[o] Text window 4
Show/Hide controls
Clear window
Set buffering limit
Log to ...
Log on/off
Clear log

Choose the [O] System option. on the Text Menu. This opens a GEM Window called System on the Backdrop. (The [O] is not part of this option's name, nor of the Window's name: it is a marker which can change to [*], as will be explained below in the section on Logging Text Windows.)

This Window, like all the Windows displayed on the Backdrop is an ordinary GEM window, and, using the mouse, you can move, resize, close and scroll it. Click in the top right-hand corner of the Window to extend it to its full size.

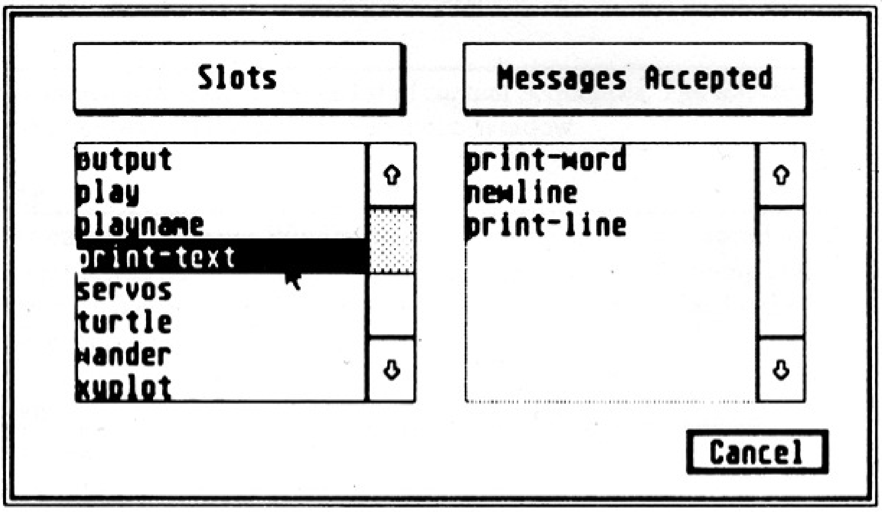
All the Text Windows can be used by Robokit processes for text output; the System Window is intended for the reporting of Robokit system information by its Slots - the Turtle Slot, for example, has reported that the Turtle is initialised in Graphics Window 1 (more of that later, in the Graphics section below). Other Slots will print their own reports and requests in the System Window from time to time. This need not prevent you from using the System Window for your own output, but it does mean that your output may be more or less randomly sprinkled with System reports.



Send a Verbose message to the Motors Slot. This instructs the Slot to print reports of its activities in the System Window. Most Slots accept this message and behave this way. Now send a few messages to Motors, and observe the results in the System Window.

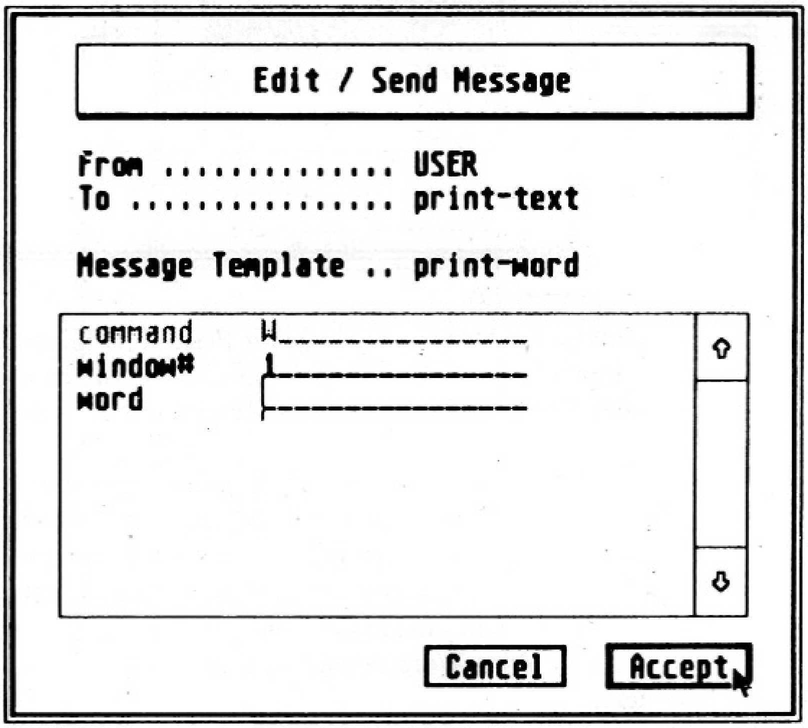
This reporting can be extremely useful when you're debugging a project, but it does slow Robokit execution quite markedly, so don't use it when speed and synchronisation are important. Sending a second Verbose message will turn this facility off. Do that now, make sure that you see the appropriate message in the System Window, then close that Window by clicking in the top left-hand corner, and open Text Window 1 from the Text menu.

We are going to Send Messages, so open the Log Window by choosing Log and Show Window on the Sequencer menu. All the Text Windows are controlled by messages to a Slot called Print_text, so choose Send on the Message menu.


Slots
output
play
playname
print-text
servos
turtle
wander
xyplot
Messages Accepted
print-word
newline
print-line
Cancel

Click on the Print_Text Destination. This Slot accepts only three messages: they enable you to print text (with and without an automatic end-of-line), and to print a newline in any Text Window - whether it is open and displayed on the Backdrop or not.

Choose the Print_Word message.



Edit / Send Message

From .... USER
To .... print-text

Message Template .. print-word

command W
window# 1
word

The Edit Message Box requres you to type in the number of the Window to which you want Text sent, and the Text to be printed there; the default is Text Window 1, as you see. The System Window number is zero.

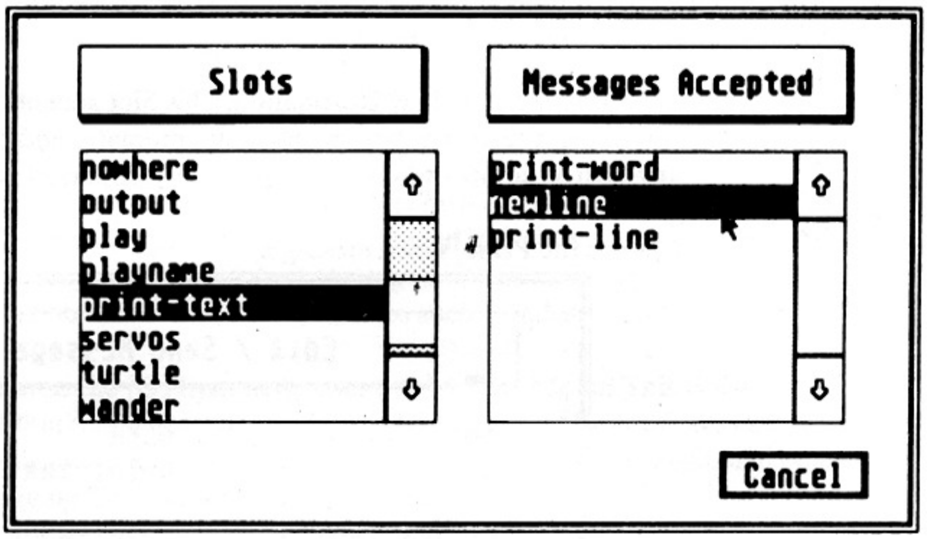
The Text which you enter (maximum of 16 characters, can be more than one word and can contain spaces and other non-alphabetic characters) is printed in the designated Text Window at the current cursor position. A space is printed after the Text, and the Window cursor remains at that position on the line. Enter any Text you like. When you Accept the Edited message your Text is printed at the cursor position in the Text Window, and the Print_Word message is recorded in the Log.

Click on the Log entry, and your Text is printed again in the Text Window, with a space separating it from the previous Text.

Double-click on the Log entry, and change your Text in the Edit Message Box. When you Accept the Edited message your new Text is printed in the Window at the cursor position.

Carry on clicking in the Log Window until your Text output reaches the right-hand edge of the Window; if the Window is at its maximum width, your Text will wrap from one line to the next at this point. Drag the bottom right-hand corner of the System Window to the left with the mouse, and notice that it has no effect on the Text in the Window - the Window is always justified with respect to its maximum possible width, not to its current width.

Choose Send on the Message menu.



Choose the Newline message for Print_Text this time, and Accept it. Nothing seems to happen in the Text Window, but click on the Print_Word entry in the Log again, and your Text will be printed in the Window at the start of a new line.

Once again, choose Send on the Message menu. Choose the Print_Line message for Print_Text. Enter your Text as before. When you Accept the message your Line Text is printed at the cursor position. Now click on the Print_Word entry in the Log, and your Word Text is printed at the start of the line after the end of the Line Text. Click on the Print_Line entry in the Log, and your Line Text is printed directly after the Word Text. Click on Print-Word in the Log again, and the Word Text is printed at the start of a new line.

When you have several lines of Text displayed in the System Window, use the mouse to drag the bottom edge of the Window above the bottom line of Text on the Window. The Window display shows the top of the Window as it was before, and the scroll bars now become active. Scroll down the Window, and the most recently output Text comes into view at the bottom of the Window.

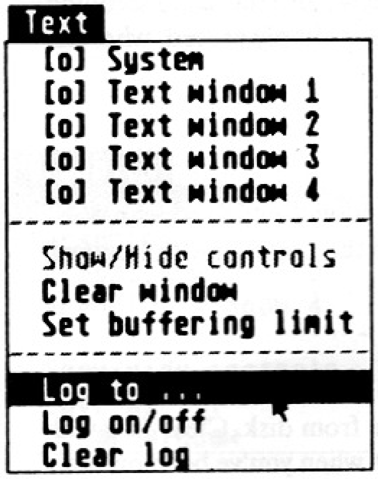
Scroll the Window down until the last Text that you output is on the top line of the Window, then click on a Print_Word or Print_Line entry in the Log. Your text is printed at the System Window cursor position. Now scroll to the top of the Window, and again click on the same Log entry. Nothing happens in the System Window, but when you scroll to the bottom of the Window, you will see that your Text has been printed in its proper place after all your other Text.

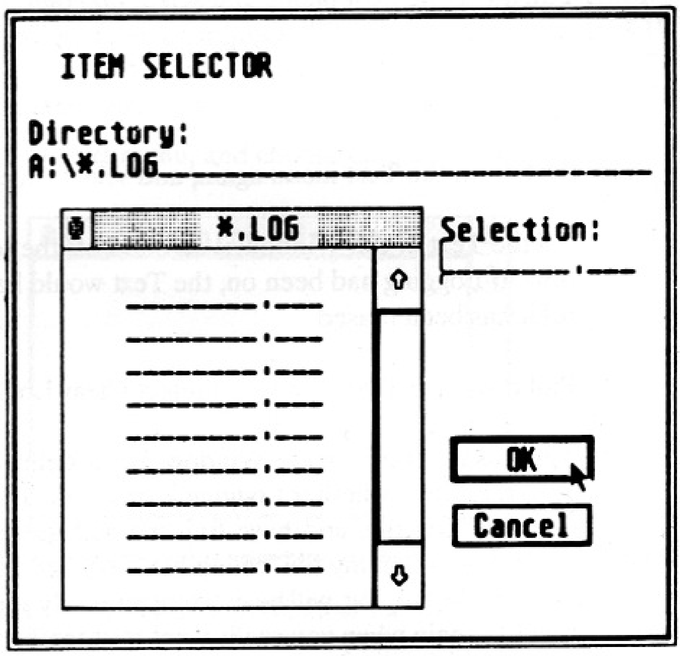
If you carry on adding Text to the bottom of the Window (the only place that you can add it) while scrolled to the top of the Window, nothing visible happens, but you will eventually see the top line of Text in the Window scroll off the top of the Window. This happens when the number of lines of Text output to the Window exceeds the Buffering Limit (see below) of that Window. Text scrolled off the Window in this Window is erased, and cannot be recaptured.

Choose Text Window 3 on the Text menu. Inspect the Window - it should be empty - and close it by clicking in its top left hand corner. Now double-click a Print_Word entry in the Log (or Send a Print_Word message to Print_Text via the Message menu, if you prefer), and change the Window number in the Edit Message Box to 3; change the Text to anything you like, and Accept the message. Nothing seems to happen, but if you open Text Window 3 via the Text menu, you will see that your Text has been output to this

Window even though it was closed at the time. Robokit keeps an up-to-date record of the output to all its Windows, whether they are open or closed.

Now with Text Window 1 open and active (click on it if its scroll bar isn't drawn), pull down the Text menu, and select Log to.


Text
[o] System
[o] Text window 1
[o] Text window 2
[o] Text window 3
[o] Text window 4
Show/Hide controls
Clear window
Set buffering limit
Log to ...
Log on/off
Clear log

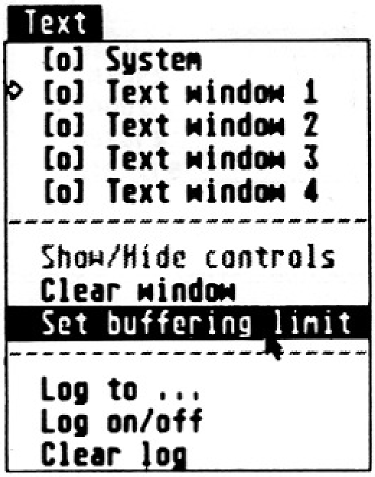

ITEM SELECTOR
*.LOG
Selection:
OK
Cancel

Directory:
A:\*.LOG

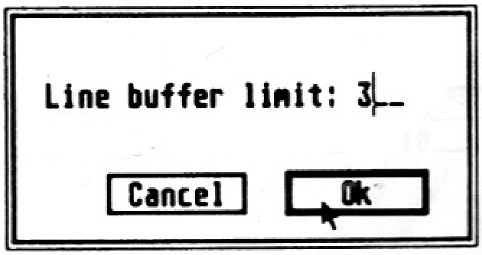
Type any file name (Robokit adds the .LOG extension). This is the name of a spool file into which Robokit will save Text output to the current Text Window. It will be a simple ASCII file which you can View on the Gem Desktop, or in a word processor.

Notice that the Text Window Title Bar now has a [*] marker in place of the [O] marker; you will see this marker on the Text menu, also. Windows marked in this way are being Logged to spool files.

Text is not sent to the Log file when it is printed in the Text Window; it is sent to the file as it scrolls off the top of the Window. Text scrolls off the top of the Window when the number of lines of Text in the Window exceeds the current Buffer Limit; the default Limit is 40 lines, which means that you have to output 41 lines of Text before one line (the first one output to the Window) scrolls off the top of the Window into the Log file.

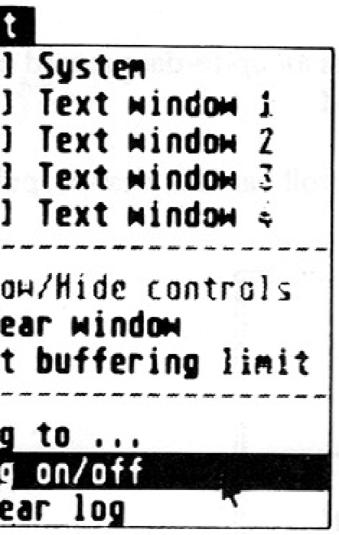

Text
[o] System
[o] Text window 1
[o] Text window 2
[o] Text window 3
[o] Text window 4
Show/Hide controls
Clear window
Set buffering limit
Log to ...
Log on/off
Clear log

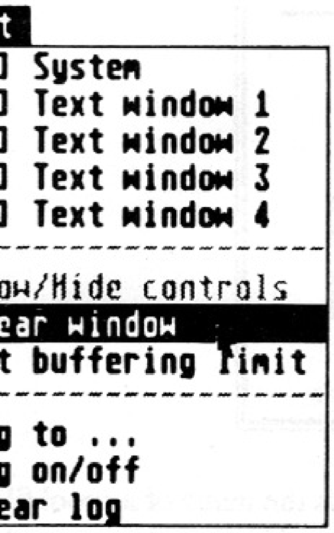
Pull down the Text menu, and select Set Buffering Limit.

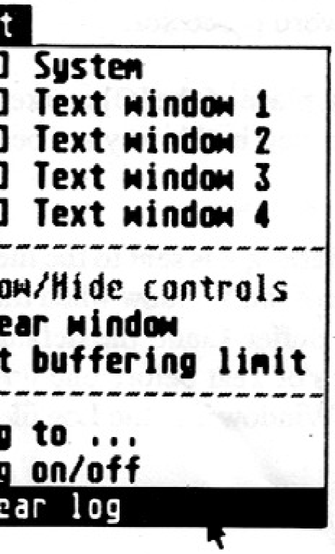


Line buffer limit: 3

You should see a pop-up box which requests the new Buffer Line Limit; enter 3, and OK it. The pop-up box closes and the disk-drive light goes on; the Text in the Window scrolls


t
] System
] Text window 1
] Text window 2
] Text window 3
] Text window 4
ow/Hide controls
ear window
t buffering limit
g to ...
g on/off
ear log


t
] System
] Text window 1
] Text window 2
] Text window 3
] Text window 4
ow/Hide controls
ear window
t buffering limit
g to ...
g on/off
ear log


t
] System
] Text window 1
] Text window 2
] Text window 3
] Text window 4
ow/Hide controls
ear window
t buffering limit
g to ...
g on/off
ear log

slowly up the Window until there are only two lines left - evrything else has been scrolled out of the Buffer and into the Log file. Print something else in this Window, and the scrolling and spooling will happen again as soon as more than three lines of Text are present in the Text Window.

Pull down the Text menu, and select Log on/off. The Logging marker on the Text Window is replaced by [O]. Output some more Text to the Window, and the same scrolling occurs, but now the Text scrolled off the top of the Wimdow is simply erased, whereas when Logging was on, the scrolled Text went into the Log file.

Pull down the Text menu again, and select Clear Window.

All the Text in the Window is scrolled off the top of the Window, leaving a clear Window. If Logging had been on, the Text would have gone into the Log file; with Logging off it has been erased.

Pull down the Text menu, and select Clear Log.

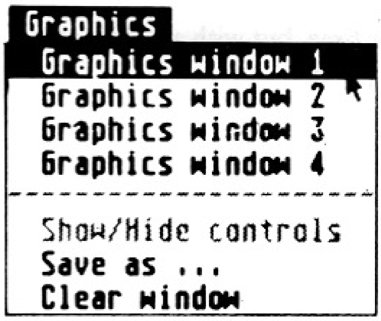
This has no effect on the Window, but it deletes the Log file from disk. Clear Log will have this effect whether Logging is on or off. This means that when you've been outputting to a Log file, and have finished with Logging, you should Log To another file - preferably a dummy file, called No_Log, for example. Now if you Clear Log, only the dummy file, No_log, will be deleted; naturally you have to remember to Log To a genuine Log file again when you really want to store Text.

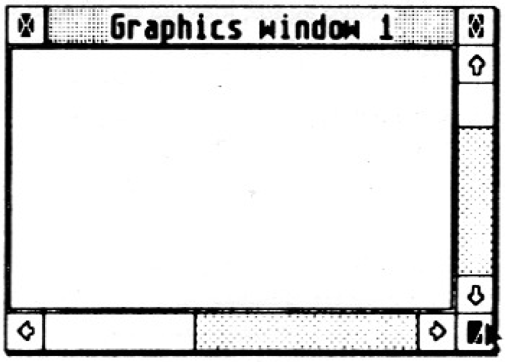
If you Log To a Log file already existing on disk, your output will be written on the end of the old file.

3.5. Graphics

Just as the Text menu provides facilities for outputting Text to various Windows, so the Graphics menu provides Graphical Window output. This can be done via two Slots - Graphics and Turtle. The Graphics Messages are a subset of the plotting and drawing commands provided by most micro-computer BASICs, while the Turtle Messages provide a LOGO subset.

Pull down the Graphics menu, and choose Graphics Window 1.

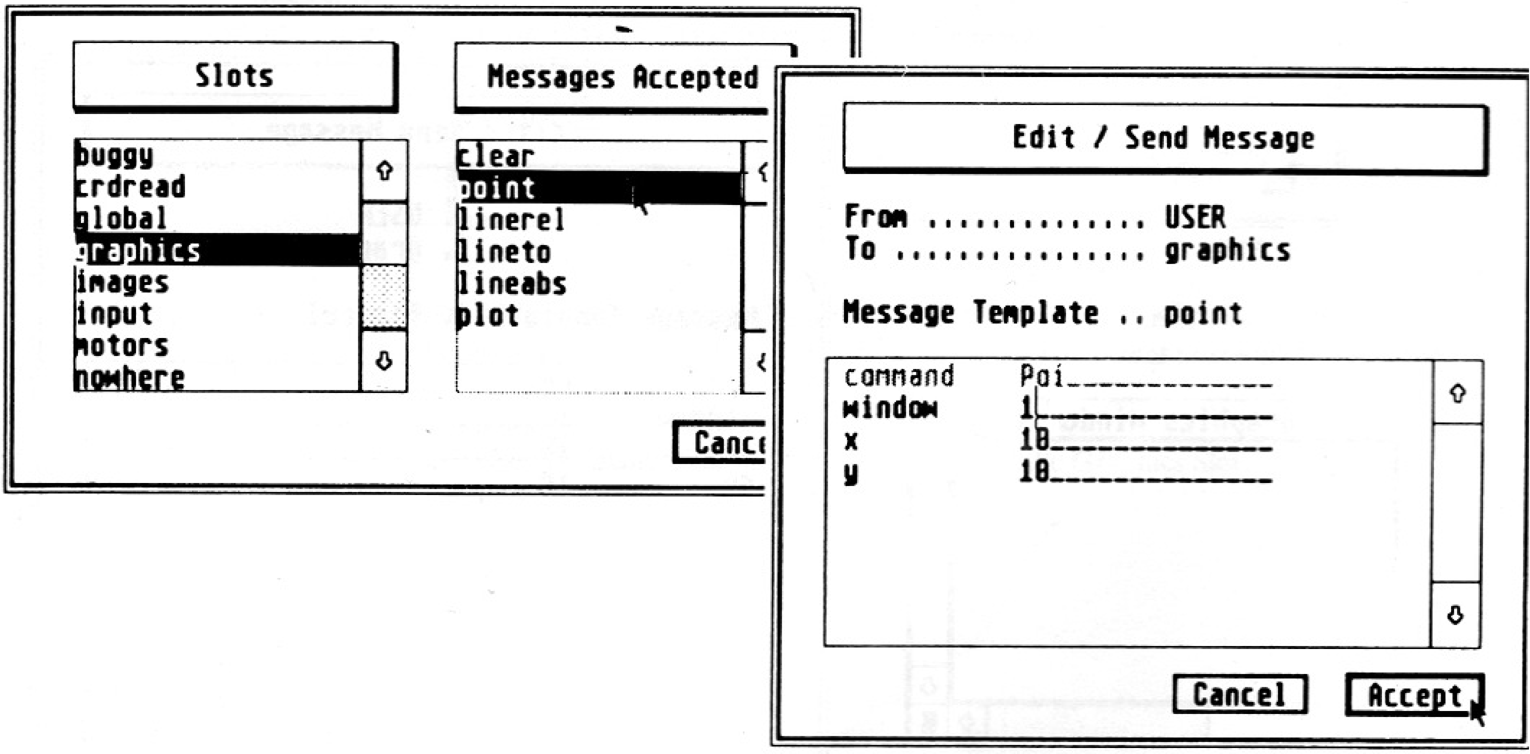

Graphics
Graphics window 1
Graphics window 2
Graphics window 3
Graphics window 4
Show/Hide controls
Save as ...
Clear window


Graphics window 1

A GEM Window entitled Graphics Window 1 appears. It can be scrolled (up and down but not side-to-side), re-sized and moved. Points on it are referred to by their X,Y co-ordinates, where X is the distance of a point from the left edge of the Window, and Y is the distance of the point from the top of the Window.

The top left corner of the Window is point (0,0); the bottom right corner of the Window is (1000,1000); the top right corner of the Window is point (1000,0); the bottom left corner of the Window is point (0, 1000)

Send a Point Message to the Graphics Slot.


Slots
buggy
crdread
global
graphics
images
input
motors
nowhere
Messages Accepted
clear
point
linerel
lineto
lineabs
plot
Canc
Edit / Send Message
Cancel
Accept

From .... USER
To .... graphics

Message Template .. point

command	Poi
window	1
x	10
y	10

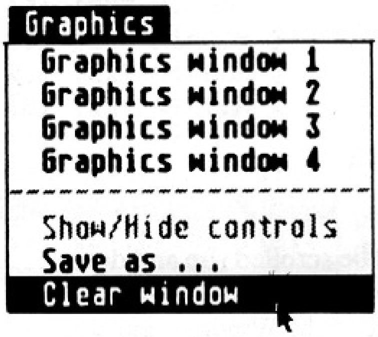
Notice that all Graphics Messages default to Graphics Window 1, so you don't need to enter anything on the Window line; choose any numbers between 0 and 1000 for the X and Y values - these are the Window co-ordinates of the point to be plotted, as explained above.

Accept the Message, and you should see a single point - the size of a full-stop - appear in Window 1; if you can't see it, try scrolling the Window.

Send the Message again, leaving the Y value unchanged, but with a different X value. The new point appears to the left or right of the previous point.

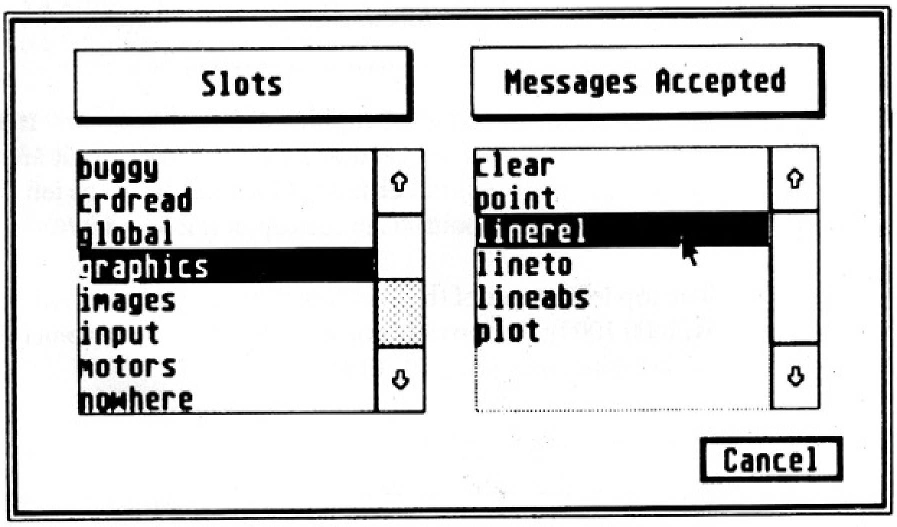
Send the Message again, leaving the X value unchanged this time, but with a different Y value. The new point appears above or below the previous point.

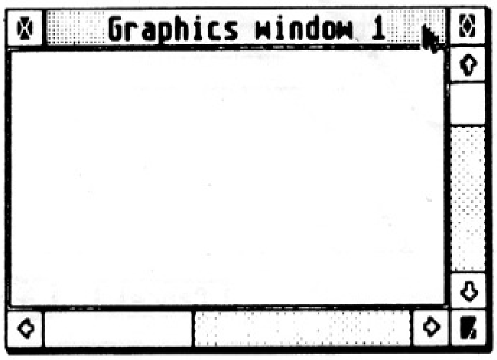
Pull down the Graphics menu, and choose Clear Window:

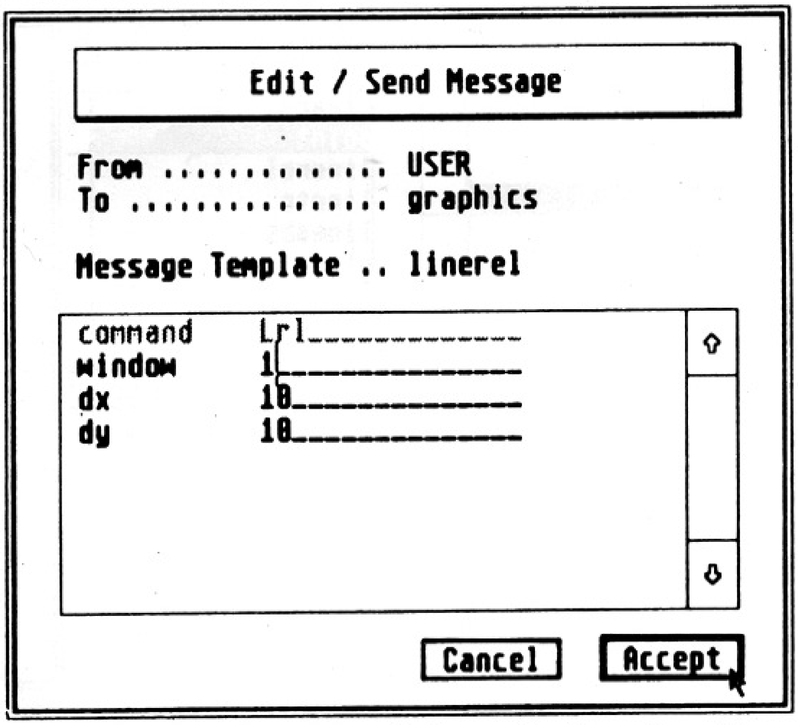

Graphics
Graphics window 1
Graphics window 2
Graphics window 3
Graphics window 4
Show/Hide controls
Save as ...
Clear window

The contents of Window 1 are erased.

Send a Linerel Message to Graphics.


Slots
buggy
crdread
global
graphics
images
input
motors
nowhere
Messages Accepted
clear
point
linerel
lineto
lineabs
plot
Cancel


Graphics window 1


Edit / Send Message

From .... USER
To .... graphics

Message Template .. linerel

command Lrl
window 1
dx 10
dy 10

Cancel
Accept

'Linerel' stands for 'Line Relative', meaning '*Draw a line from the current position of the Graphics cursor, to the point whose co- ordinates are obtained by adding Dx and Dy to the current X and Y values*'

The Graphics cursor at the moment is at the position of the last point you plotted with the the Point Message (Clearing the Window does not affect the Graphics cursor position); if you enter 10 for Dx and 20 Dy, and Accept the Message, you will see a short line in Graphics Window 1 drawn rightwards and downwards from the position of the last point plotted.

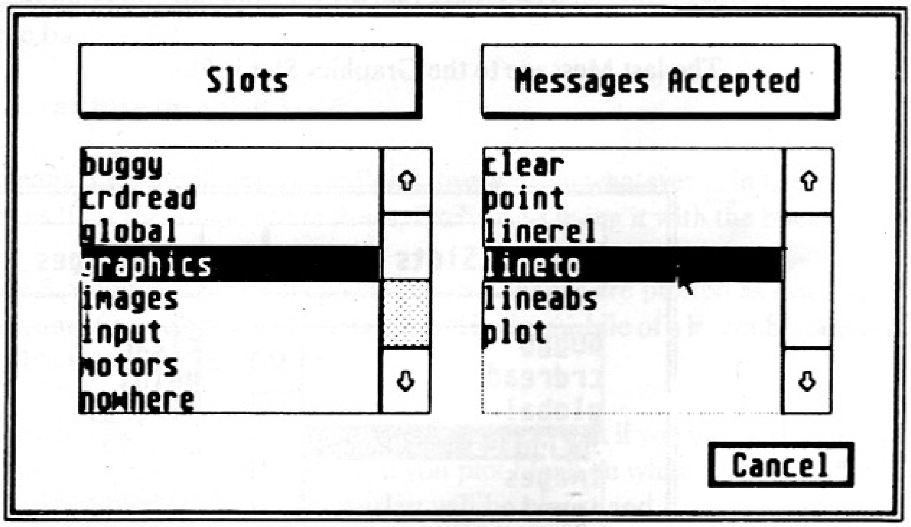
If the last point plotted was at (60,110), then the line is drawn from there to the point (60 + 10,110 + 20), ie (70,130).

Send the Linerel Message again; leave Dy unchanged, but enter 0 for Dx. A new line appears, drawn vertically down the Window from the end of the previous line.

Send the Linerel Message once again; set Dx to 20, and make Dy 0. This time a line is drawn horizontally rightwards across the Window from the end of the previous line.

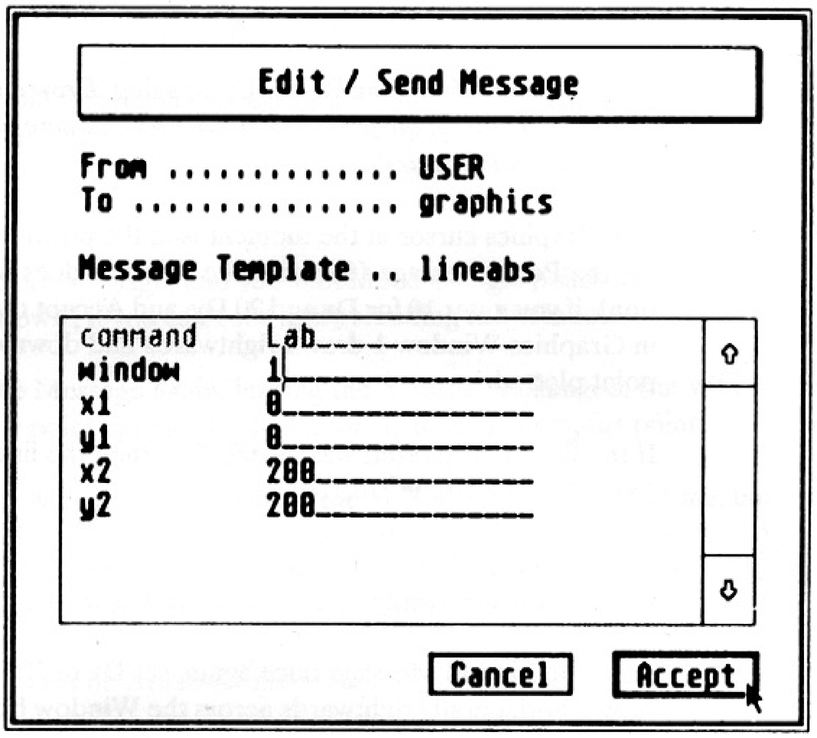
This is Relative Plotting - you never say where you are, or where you want to go; you simply say how much you want to move from wherever you are at the moment.

A slight departure from this is illustrated by the Lineto Message to Graphics.



In this Message you specify the Absolute (X,Y) position of the point to which you want to draw a line, and the line is drawn to there from the current position of the Graphics cursor.

Compare this finally with the Lineabs Message to the Graphics Slot.



Edit / Send Message

From .... USER
To .... graphics

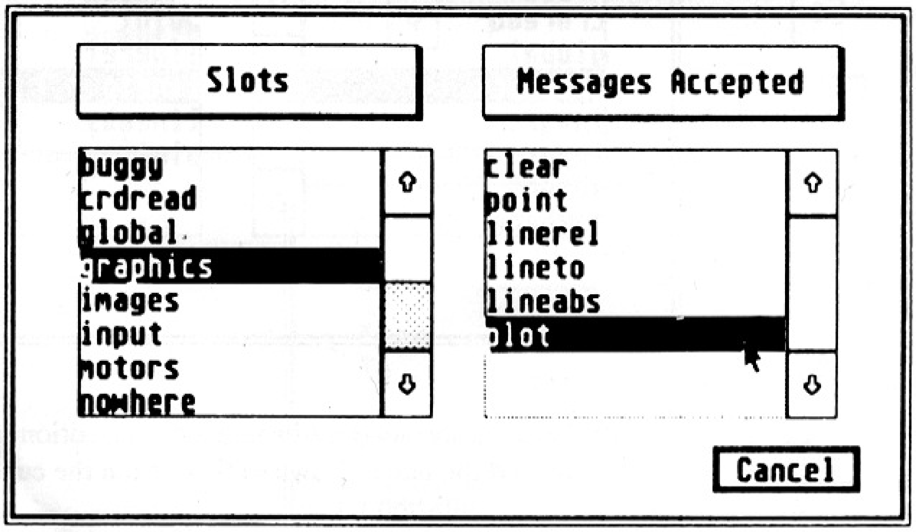
Message Template .. lineabs

command	Lab
window	1
x1	0
y1	0
x2	200
y2	200

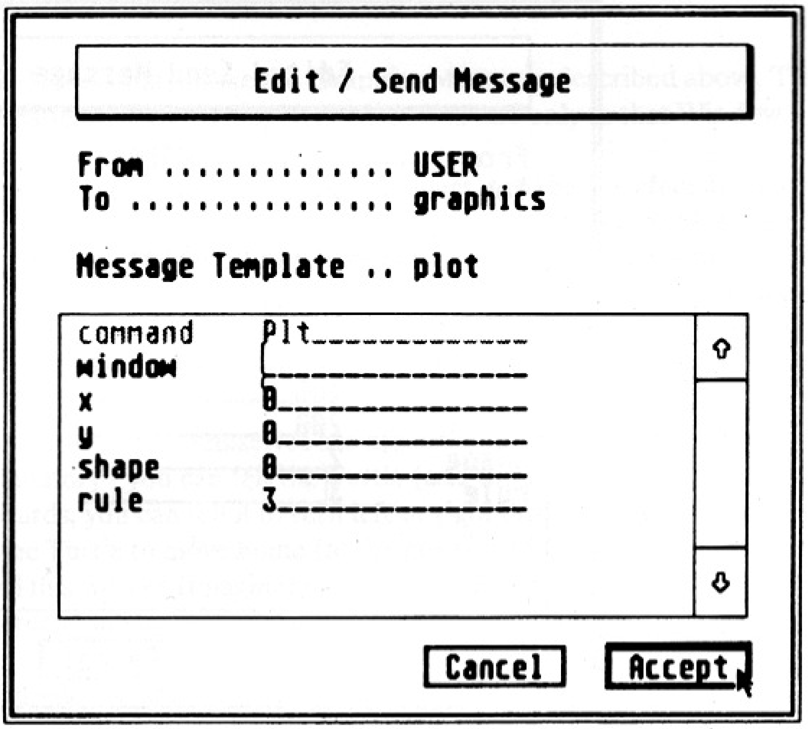
Cancel
Accept

Here you must enter (X1,Y1), the position of one end of the line, and (X2,Y2), the position of the other end. Consequently, if you Send the same Lineabs Message twice, you should see only one line drawn (since the second Message causes a line to be drawn over the top of the first line), whereas Sending the same Linerel Message twice will nearly always produce two lines, one drawn from the end point of the other.

The last Message to the Graphics Slot is Plot.



Slots
buggy
crdread
global
graphics
images
input
motors
nowhere
Messages Accepted
clear
point
linerel
lineto
lineabs
plot
Cancel



From .... USER
To .... graphics

Message Template .. plot

command	Plt
window	
x	0
y	0
shape	0
rule	3

Plot is an absolute-plotting Message, so you must enter a Window position for X and Y in the Message. Shape can have the value 0, 1 or 2:-

0 produces a point shape at (X,Y), 1 produces a small square at (X,Y), 2 produces a large ball at (X,Y).

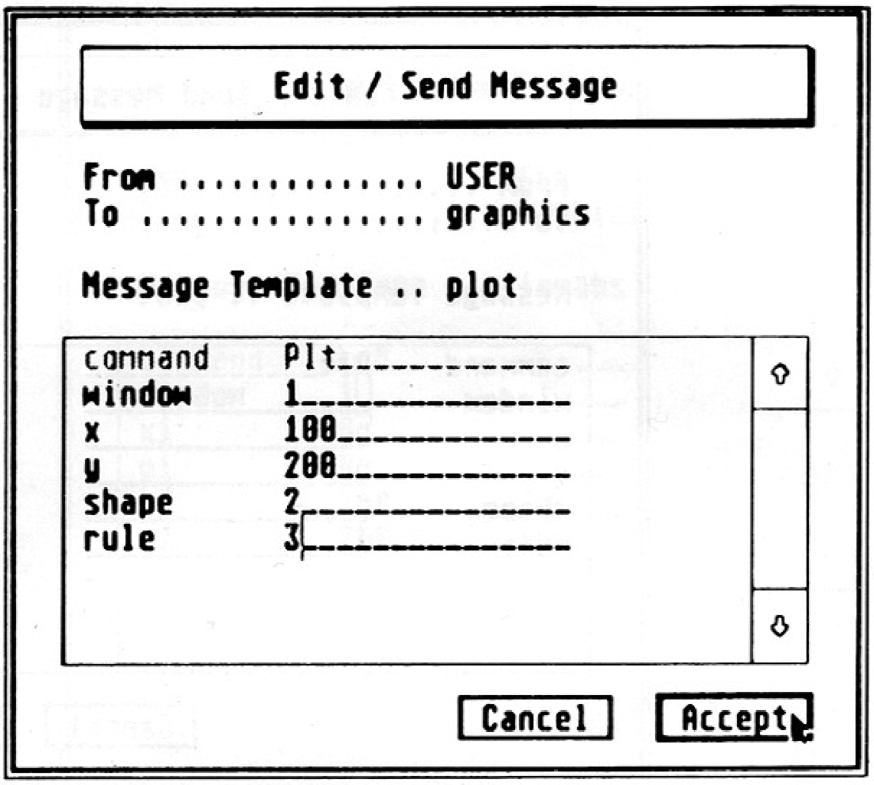
Rule can have the value 3 or 6:-

3 means 'Plot the Shape at the Point, overwriting whatever is in the Window already' 6 means 'Plot the Shape at the Point, Exclusive-Or-ing it with the background'

Rule 3 produces the results you'd expect - Shapes are plotted as instructed in the Window, and if a small black Shape is plotted in the middle of a larger black Shape, you won't be able to see the small Shape.

Rule 6 is different - the Exclusive Or rule means that if you plot a black Shape on a black Shape, the overlap will be white; if you plot black on white, the overlap will be black; if you plot white on black, the overlap will be black; and if you plot white on white, the overlap will be white. Plotting one colour on a different colour, therefore, produces black; plotting one colour on the same colour produces white. As you shall see.

Clear Graphics Window 1, and Plot a ball (Shape 2) at (100,200), with Rule 3.

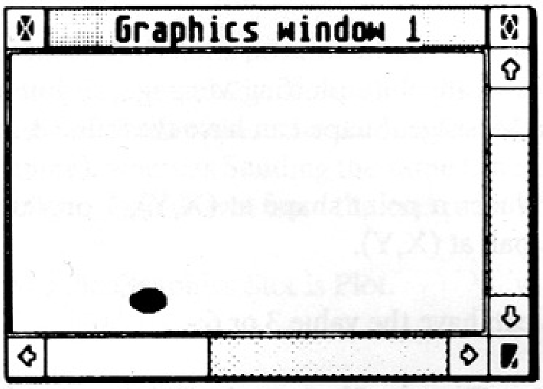


Edit / Send Message

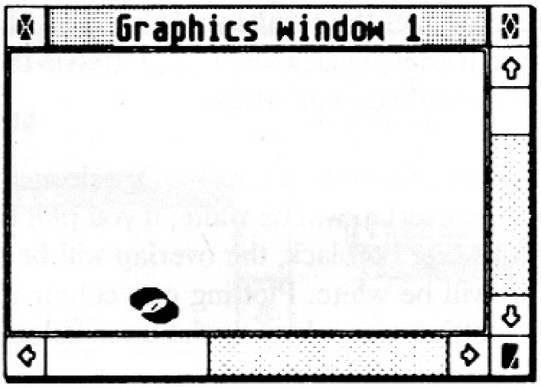
From .... USER
To .... graphics

Message Template .. plot

command	Plt
window	1
x	100
y	200
shape	2
rule	3



Now Send the Plot Message again, to Plot a ball at (110,210) with Rule 6.



The second ball overlaps the first, and the overlap is white, except for the white highlight on the ball, which is black in the overlap. This demonstrates Exclusive-Or plotting of black on black, black on white, and white on black.

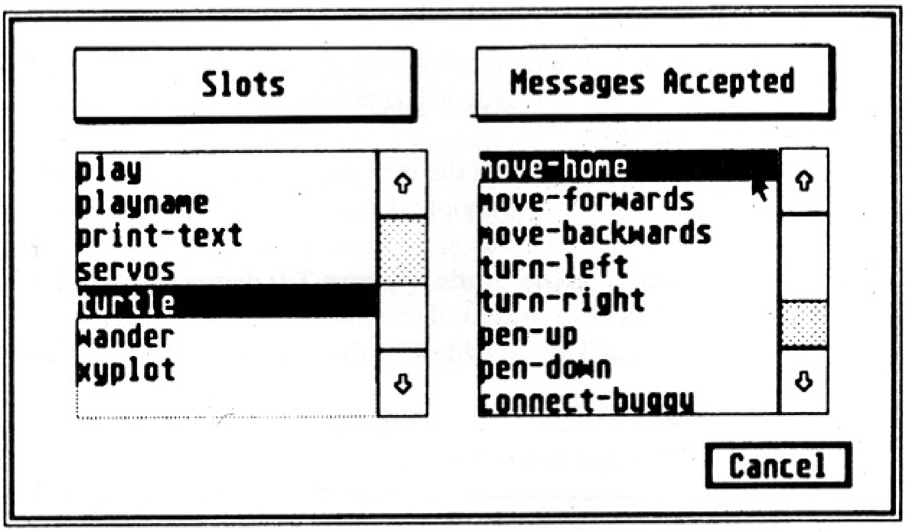
These Point and Line Messages can be used in Sequences to draw diagrams and maps to accompany the operations of your robots.

This can be done in a rather more convenient way with Turtle Graphics. Here, Robokit emulates the LOGO language, allowing you to Send Messages simultaneously to the Buggy Slot (which can drive a two-motor floor vehicle) and to the screen Turtle which mimics it. Notice that, unlike the Graphics Messages described above, Turtle Messages are automatically sent to Graphics Window 1, and only to that Window.

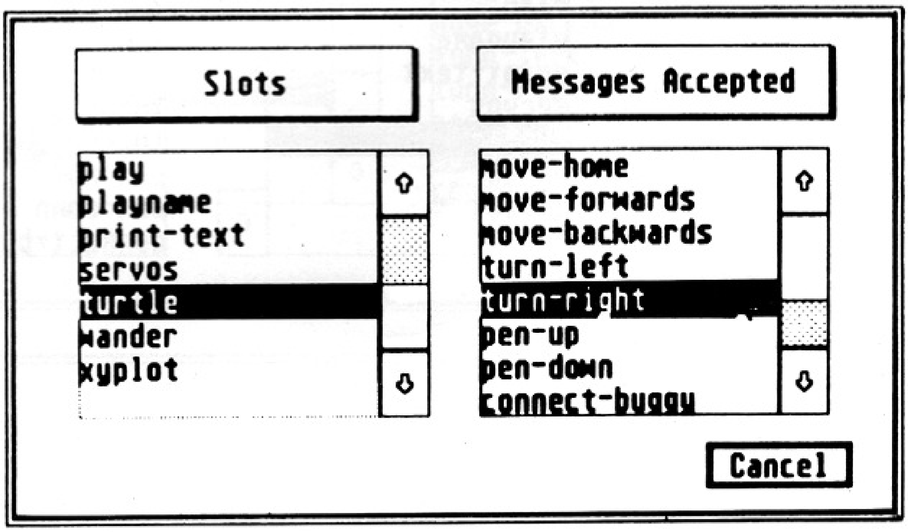
If you have built some kind of Buggy, connect it to the Interface Board in place of the Mini_Arm, and Send the Connect_Buggy Message to the Turtle Slot. (For further details look up the Turtle Slot and Buggy Slot in the Slot Reference Section.) This ensures that Messages Sent to the screen Turtle are also Sent to the Buggy; if you don't want to connect a Buggy, then don't Send this Message - all Turtle Messages will be addressed to the screen Turtle only.

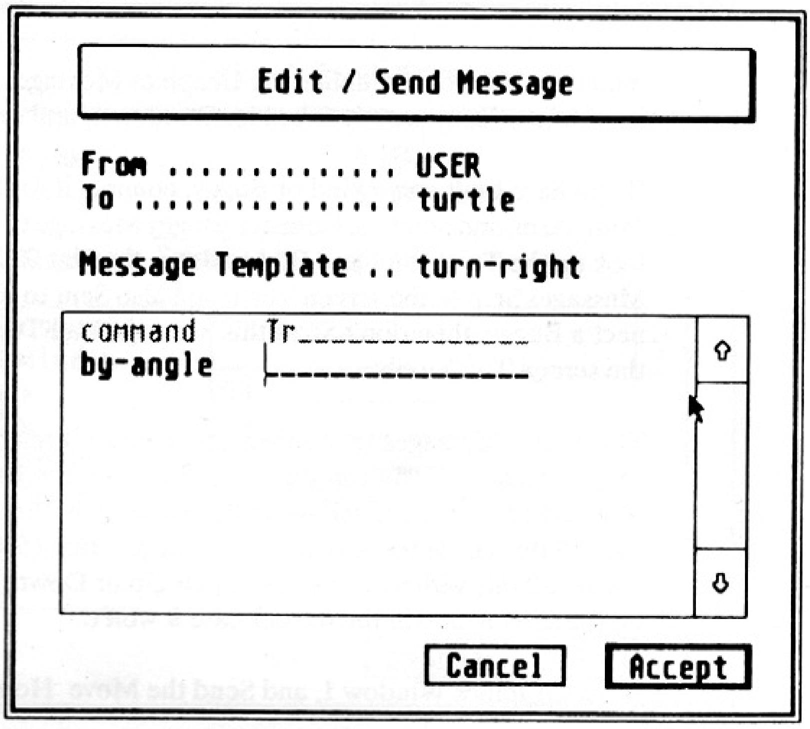
The Turtle Messages themselves are almost all relative plotting Messages, and largely self-explanatory - you can tell the Turtle (and the Buggy) to move a distance forwards or backwards, you can tell it to turn left or right through a number of degrees, and you can tell the Turtle to move home (to the position (500,500) in the Window). The Turtle can do all this with its (imaginary) pen Up or Down; in the latter case, it will draw lines on the screen, and in the former case it won't.

Clear Graphics Window 1, and Send the Move_Home Message to the Turtle Slot.


Slots
Messages Accepted
play
playname
print-text
servos
turtle
wander
xyplot
move-home
move-forwards
move-backwards
turn-left
turn-right
pen-up
pen-down
connect-buggy
Cancel

The Turtle (a small square, just like Shape 2 described in the Graphics section above), moves to (500,500) in Graphics Window 1, and faces North (up the screen). Send a Turn_Right Message.


Slots
Messages Accepted
play
playname
print-text
servos
turtle
wander
xyplot
move-home
move-forwards
move-backwards
turn-left
turn-right
pen-up
pen-down
connect-buggy
Cancel



Edit / Send Message

From .... USER
To .... turtle

Message Template .. turn-right

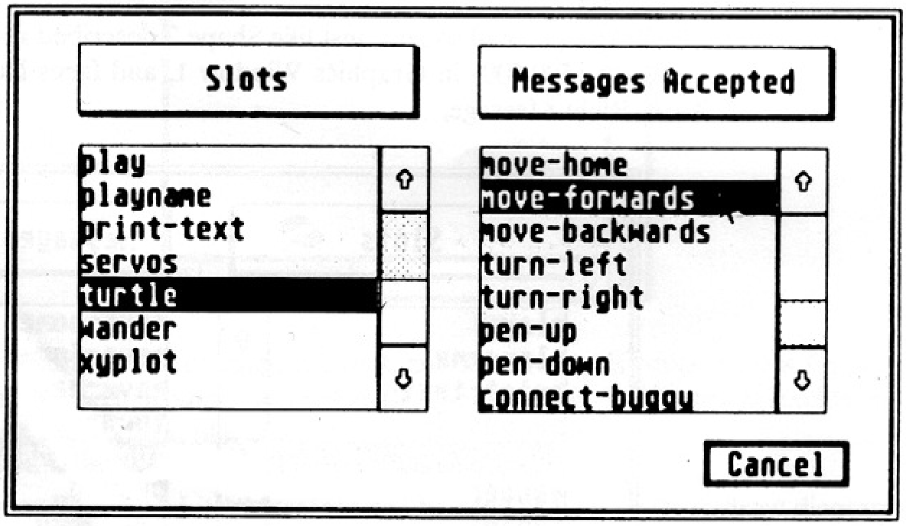
command Tr
by-angle

Cancel
Accept

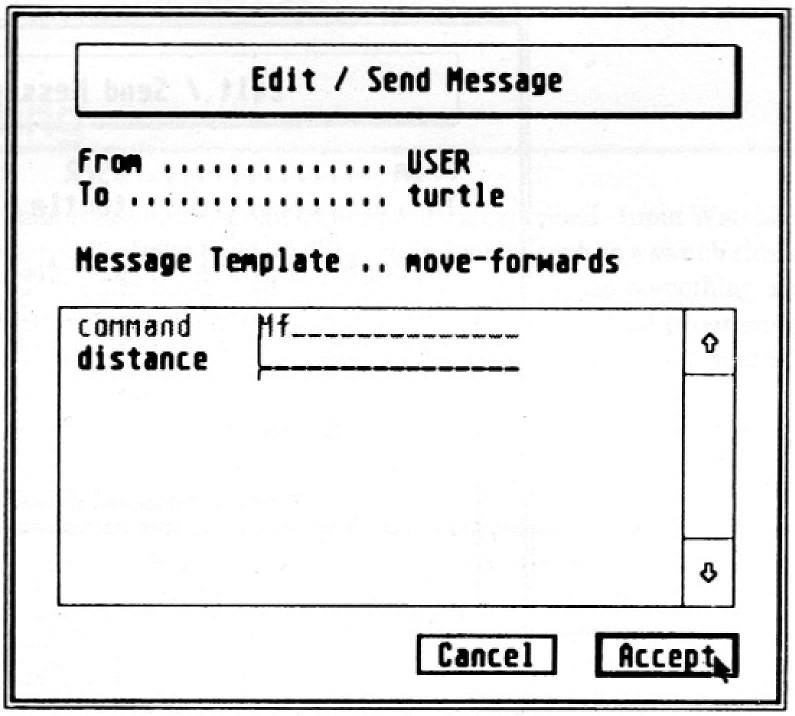
Enter an angle (in degrees) through which the Turtle is to turn; this is a Turn_Right Message so the Turtle will rotate to the Right, ie Clockwise.

If you enter 90 as the turn angle, then the Turtle will turn right, as we commonly understand that instruction; if you enter 180, then the Turtle will turn 180 degrees to the right, which is the same as doing an about-turn, and facing the opposite direction; if you enter 270, then the Turtle will turn 270 degrees to the right, and thereby finish up facing 90 degrees to the left; if you enter a negative turn angle, the Turtle turns the opposite way, so that Turn_Right -37 is the same as Turn_Left 37, for example.

When you accept the Message, nothing happens on the screen. Send a Move_Forwards Message to the Turtle.



Slots
play
playname
print-text
servos
turtle
wander
xyplot
Messages Accepted
move-home
move-forwards
move-backwards
turn-left
turn-right
pen-up
pen-down
connect-buggy
Cancel


Edit / Send Message

From .... USER
To .... turtle

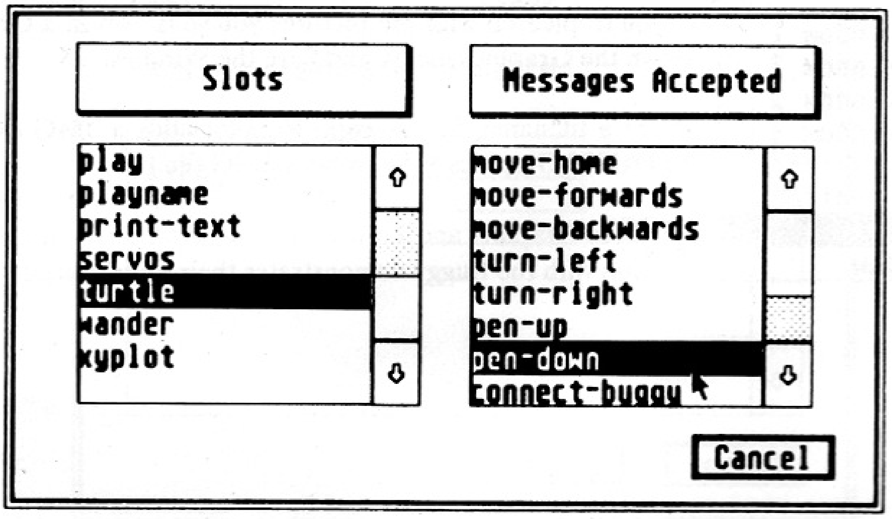
Message Template .. move-forwards

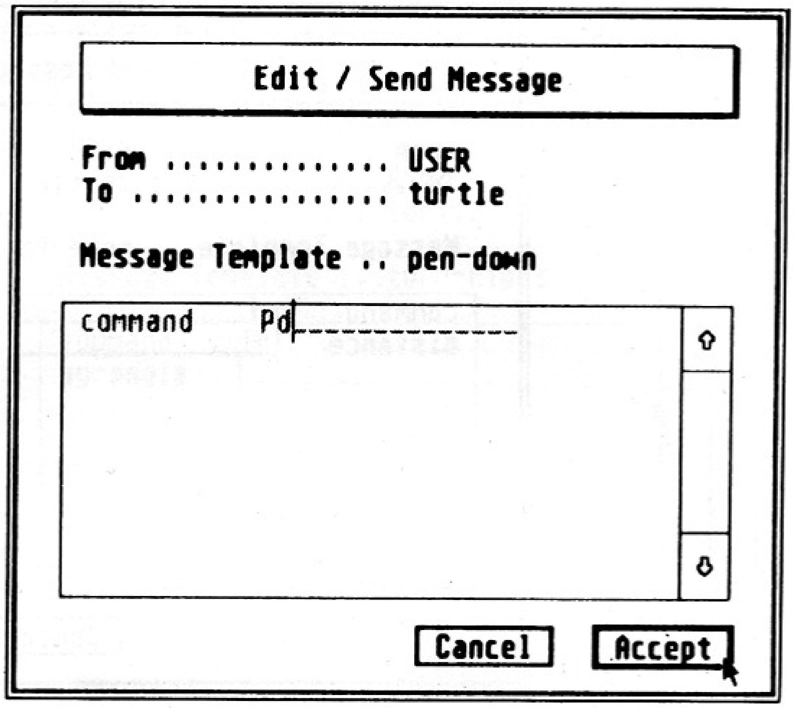
command Mf
distance

Cancel
Accept

Enter a distance; the Turtle moves pixel-by-pixel in Graphics Window 1, which measures 1000 by 1000. When you Accept the Message, the Turtle moves as directed. If you send the Turtle to a position in which one or both of the co-ordinates are greater than 1000 or less than 0, it will be invisible but will preserve its position, and will apply new Messages to that position. If you Move the Turtle Forwards from (900,200) to (1100,200), for example, the Turtle will be invisible; if you then Send a Move_Backwards 500 Message, the Turtle will Move_Backwards to (1100-500,200)(600,200).

You can see the Turtle on the screen, but you can't see where it's been, so Send a Pen_Down Message.


Slots
play
playname
print-text
servos
turtle
wander
xyplot
Messages Accepted
move-home
move-forwards
move-backwards
turn-left
turn-right
pen-up
pen-down
connect-buggy
Cancel



Edit / Send Message

From .... USER
To .... turtle

Message Template .. pen-down

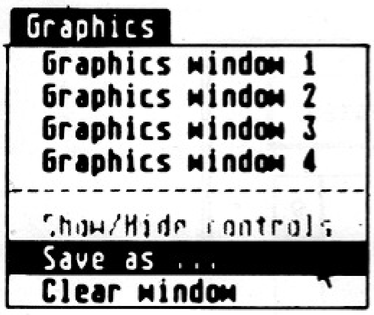
command Pd

Cancel
Accept

Nothing happens on the screen, but if you Send the Move_Forwards Message again, the Turtle will move forwards, drawing a line.

Send the Turn_Right Message again, followed by the same Move_Forwards Message, and the Turtle moves again on a new heading, demonstrating that the Turn Messages are relative rather than absolute - the Turtle turns through the specified angle from its current heading.

Send the Move_Backwards Message, and you'll see that the Turtle does exactly what you'd expect - it moves backwards, but continues to face forwards (as it were: its position changes, but not its heading).



Graphics
Graphics window 1
Graphics window 2
Graphics window 3
Graphics window 4
Show/Hide controls
Save as ...
Clear window

If you're pleased with the pictures you've drawn in a Graphics Window, you can pull down the Graphics menu, and Save the Window.

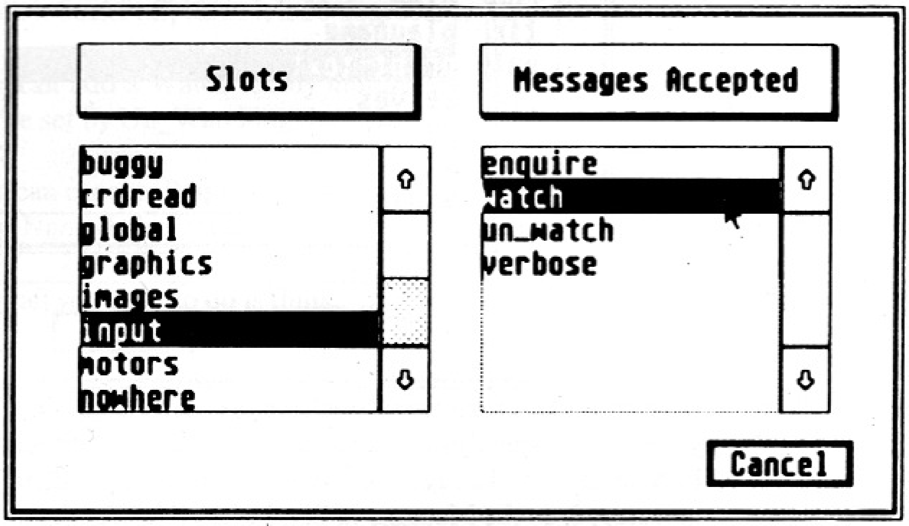
Enter a filename, and Accept; Robokit adds a .IMG extension, and saves the entire 1000x1000 Graphics Window as a .IMG file.

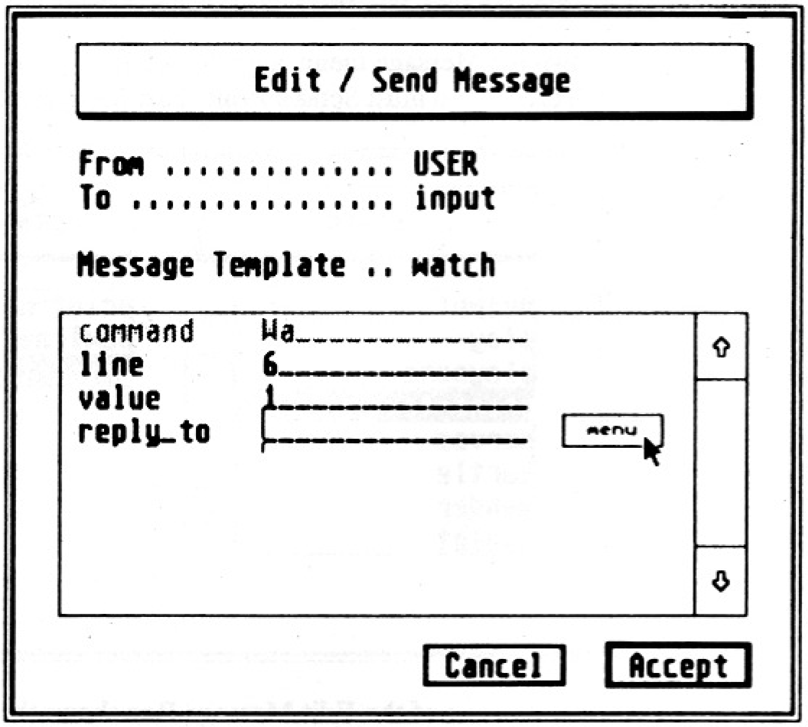
All these Graphics and Turtle Messages can be used in Sequences, and the Turtle's connection with the Buggy demonstrates their initial purpose.

3.6. Input Watching

There remains just one Robokit technique to be explained - Input Watching. This enables you to instruct Robokit to watch for certain events (such as a switch closing because the Buggy has hit something, for example), and then to do something about it. This is equivalent to the IF..THEN statement in most conventional programming languages, but rather more powerful, since once you've instructed Robokit to Watch for something, it will do so five times per second until told to stop Watching; when the event it's Watching for occurs, Robokit performs the consequent action immediately.

Send a Watch Message to Input.


Slots
buggy
crdread
global
graphics
images
input
motors
nowhere
Messages Accepted
enquire
watch
un_watch
verbose
Cancel


Edit / Send Message

From .... USER
To .... input

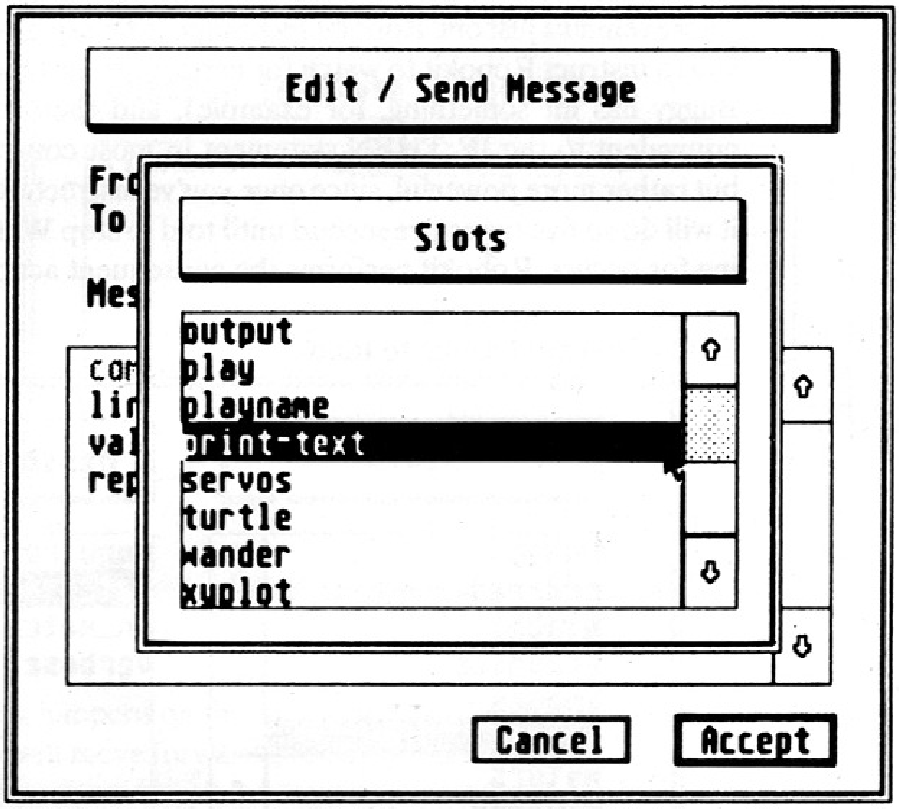
Message Template .. watch

command Wa
line 6
value 1
reply_to

menu
Cancel
Accept

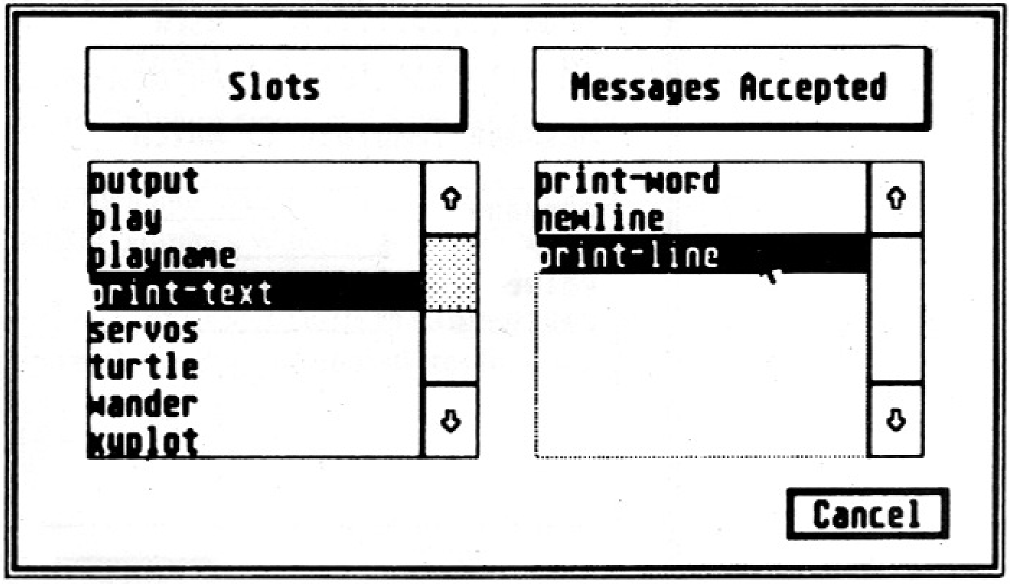
Enter the number of an Interface Input Line that's not being used by anything - say Line 6. Robokit will Watch this Line, waiting for it to attain the Value that you enter; here, a Value of 1 means that the Line has been connected to Ground, and this is the simplest event to demonstrate, so enter 1.

Reply_To means the name of the Slot to be notified when the Watched-for event occurs. If you click on the small box marked Menu, you'll see a selector box listing all the Robokit Slots by name; click on Print_Text.


Edit / Send Message
Fr
To
Mes
con
lin
val
rep
Slots
output
play
playname
print-text
servos
turtle
wander
xyplot
Cancel
Accept

You've told Robokit what to Watch for, and which Slot to notify when it happens, now you've got to specify the actual Message that Robokit is to Send to that Slot. You do this by Sending the Message itself, now. The Input Slot will intercept the next Message Sent, and store it as the Message to Send when this Watch comes true.

Pull down the Message menu, and choose Send. You've decided that Watch is to notify Print_Text, so you must Send a Print_Text Message; choose Print_Line.


Slots
output
play
playname
print-text
servos
turtle
wander
xyplot
Messages Accepted
print-word
newline
print-line
Cancel

As usual, the top of the Edit Message Box shows the source (USER) and destination (Print_Text) of the Message; for once, it's wrong. Move the cursor onto the To line of the Box, and delete Print_Text. Replace it with Input. Now a Message which is acceptable only to the Print_Text Slot will be Sent to the Input Slot; the Input Slot will accept it (though not obey it), recognising that this is the Message to Send to Print_Text, when the Watch comes true.

Now enter some Text on the Text line, and Accept the Message. Nothing happens on screen. Open Text Window 1, and Clear it. Insert one end of a piece of wire into Input Line Terminal 6 (or whichever you chose) on the Interface, and touch the other end of the wire to the 0v Terminal; this has the effect of sending Input Line 6 to the 1 state, making the Watch come true. You should see your Message output to Text Window 1.

At last, you can build real robots - you've just closed the feedback loop, and turned your remote- controlled automata into real robots - autonomous programmable physical devices with conditional branching on input. Hitherto your robots did what you told, when you told them; now you can tell them what to do, when to do it, when to do something else, and what to do when something happens. That's conditional branching on input. It means that your Buggy can potter about on the floor until it hits something, on which it backs off, turns left 30 degrees, and goes back to pottering.

You can add a Watch to any number of Input Lines, and you can change any Watch you've set by Un_Watching the Line, and adding a new Watch.

You can execute Sequences from a Watch by Replying_To Playname, then Sending the Play_Named_Seq Message to Input, just as you did with Print_Text and Print_Line.

Now all you have to do is think.

4. Robokit Slot Reference Section

This reference section gives a brief overview of each slot available in the system, along with a detailed description of the messages received by each slot and their functions.

A general overview of the operation of slots and the way in which they communicate can be found in the software guide. Throughout this documentation the word 'click' is used to signify some form of input pulse to the computer. This pulse can take the form of a switch pulling the input line low or a pulse obtained from a shaft encoder.

Table of Contents

4.1 Output Slot

4.2 Input Slot

4.3 Motors Slot

4.4 Servo Slot

4.5 Buggy Slot

4.6 Wander Slot

4.7 Playname Slot

4.8 Play Slot

4.9 Card Reader Slot

4.10 Images Slot

4.11 Turtle Slot

4.12 Graphics Slot

4.13 X-Y Plot Slot

4.14 Print-text Slot

4.1. Slot Name : OUTPUT

Description: Performs all output to the interface allowing the setting, resetting and inverting of any line. The slot also performs toggling of output lines for user-defined on and off periods. In all messages in which the output line is specified the line must be within the range of 0 to 7, the number of permitted output lines.

Set_line:

Operation to set the output line specified by 'line' to high.

Field name	Meaning	Value range / Text
Command	Operation command	'Se'
line	Output line number	0 - 7

Reset_line :

Operation to reset the line specified by 'line' to a low value.

Field name	Meaning	Value range / Text
Command	Operation command	'Re'
Line	Output line number	0 - 7

Invert_line :

Operation to invert the output line value currently on the line specified by 'line'.

Field name	Meaning	Value range / Text
Command	Operation command	'In'
Line	Output line number	0 - 7

Toggle_line :

Toggles the specified output line between high and low for the time periods specified by the parameters 'on_time' and 'off_time' respectively. On and off times are in tenths of seconds giving a maximum on or off period of 54 minutes.

Field name	Meaning	Value range / Text
Command	Operation command	'To'
Line	Output line number	0 - 7
On-time	Line high period	0 - 32767
Off-time	Line low period	0 - 32767

Toggle_off :

Stops toggling the specified line on and off by removing it from the toggle list.

Field name	Meaning	Value range / Text
Command	Operation command	'Rt'
Line	Output line number	0 - 7

Verbose:

Switches the slot into verbose reporting mode which causes messages to be printed in the system window to keep the user more fully informed as to the internal status of the slot. Sending this message to the slot acts as a toggle switch turning verbose reporting to the opposite of its present status. Operation of the slot is slowed down by enabling this option due to the outputting of text, so in time dependant applications it may be necessary to have verbose reporting disabled.

Field name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Input slot.

4.2. Slot Name : INPUT

Description: This slot provides facilities to read the value of an input line and to constantly monitor an input line. When monitoring for a specific value the slot will wait for the value to arrive and will then take some pre-defined action when this value arrives.

Enquire:

Operation to read the value from the input port of the specified line number. The Input slot replies with a message containing the read value to the slot specified in the 'reply_to' field of the enquire message.

Field name	Meaning	Value range / Text
Command	Operation command	'En'
Line	Input line number	0 - 7
Reply_to	Slot name for reply	Slot address

_rply:

This message is used as a reply by the Input slot and is sent to the slot specified in the 'reply_to' field of the enquire message. It contains the current value of the input line specified in the Enquire message.

Field name	Meaning	Value range / Text
Command	Operation command	'Ir'
Line	Input line number	0 - 7
Value	Input line value	0 or 1

atch :

Operation to monitor an input line for a specified value. Upon recognition of the requested value a user defined message is sent to the slot specified in the watch message. The reply message is set up as follows. Immediately after receiving a 'watch' message the Input slot expects to receive another message which it will store away and use as the reply message when the requested value appears on the input line.

Field name	Meaning	Value range / Text
Command	Operation command	'Wa'
Line	Input line number	0 - 7
Value	Input value to wait for	0 or 1
Reply_to	Slot to send reply to	Slot name

watch:

Operation to remove a specified line number from the list of input lines being monitored for a specific value.

Field name	Meaning	Value range / Text
Command	Operation command	'Uw'
Line#	Input line number	0 - 7

rbose:

Switch verbose reporting on or off. This reporting mode outputs informational text messages to the system window to keep the user more fully informed as to the status of the slot. Repetitive sending of this message to the input slot will toggle verbose reporting between on and off. Operation of the slot is slowed down by enabling this option due to the outputting of text, so in time dependant applications it may be necessary to have verbose reporting disabled.

Field name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Output slot.

4.3. Slot Name : MOTORS.

Description: Performs all motor control providing speed and direction control along with commands to drive a motor for a specified distance, time period or number of input clicks from an encoder. In order to provide the click and distance functions, messages are provided to allow the association of a shaft encoder with a specific motor, giving feedback for motor control. A facility is also provided to send a response message to a user specified slot when a motor has completed a requested move, be it for time, distance or clicks. Motors occupy two output lines each in order to provide direction control. It is therefore only possible to drive four motors at once from the output port, these being numbered 0 - 3. The speed of motors can be varied within a range of five settings (0 - 4), speed 0 is stopped and speeds 1 - 4 provide increases in speed of 25% each.

Init_motor :

Operation to initialise a motor on a specified output line. The control of a motor occupies two output lines in order to provide direction control, so it is important to remember that when specifying the output line number in the 'init_motor' message the slot will allocate the line specified along with the following one for control of that motor. It is therefore impossible to set up a motor on output line seven because this is the last output line and the slot will be unable to allocate line eight for direction control. When initialising several motors it is important to remember to initialise each motor on a line number which is numerically two higher or lower than any of the other motor line numbers. This is to avoid overlapping of control lines for motors.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mi'
Motor#	Motor number	0 - 3
Output#	Output line number	0 - 6

Remove_motor:

Operation to remove an initialised motor from the output port.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mr'
Motor#	Motor number	0 - 3

Move_motor :

Operation to set a motor in motion at a specified speed and in a specified direction.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mv'
Motor#	Motor number	0 - 3
Speed	Speed of motor	0 - 4
Dir	Direction of motor	1 - Forwards -1 - Backwards

Move_time :

Operation to run a motor for a pre-defined period of time at a set speed and direction. Upon completion of the move a reply message is sent to the slot specified in the 'move_time' message. The message can be either a default or a user-defined one. The default completion message takes the form of two characters 'M!' followed by the number of the motor concerned. If no reply slot is specified in the 'move_time' message then no reply will be sent. The time period for the motor to run is measured in tenths of seconds.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mt'
Motor#	Motor number	0 - 3
Speed	Speed of motor	0 - 4
Dir	Direction of motor	1 - Forwards -1 - Backwards
Time	Duration of move	0 - 32767
Reply_to	Slot to reply to	Slot name

Move_dist :

Operation to move a motor for a specified distance at a set speed and direction. This operation only functions correctly if an encoder has been initialised and calibrated for the motor in the 'move_dist' message. If a reply slot is specified then either a user defined or default reply message will be sent. The default message takes the form of the two characters 'M!' followed by the motor number.

Field Name	Meaning	Value range / Text
Command	Operation command	'Md'
Motor#	Number of motor	0 - 3
Speed	Speed of motor	0 - 4
Dir	Direction of motor	1 - Forwards -1 - Backwards
Dist	Distance of move	0 - 32767
Reply_to	Slot to reply to	Slot name

[illegible]

[illegible]

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
		[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

[illegible]

[illegible]

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

Set_absolute:

Operation to set the absolute position of the motor. This operation can be used to set the current position of a motor to any value. This is particularly useful for setting the origin for the motor, where zero is passed as the new absolute position. The 'set_absolute' command need not be called before making relative moves, by default the origin for a motor is taken as the position of the motor upon the first use of the 'move_absolute' message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Sa'
Motor#	Motor number	0 - 3
Position	New position on scale	-32768 - 32767

Motor_stop :

Operation to stop a motor.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ms'
Motor#	Motor number	0 - 3

Encoder_init:

Operation to initialise an encoder on a specified input line and for a specified motor. This message must be sent to the motors slot before any of the operations requiring encoder feedback for a motor are used.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ei'
Motor#	Motor number	0 - 3
Inp_line	Input line number	0 - 7

Enc_calib :

Operation to calibrate an encoder for measuring the distance travelled by a motor. The slot records how many clicks are expected for the specified distance and subsequently uses this information for any of the slot operation which involve measuring the distance travelled by a motor.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ec'
Motor#	Motor number	0 - 3
Dist	Distance	0 - 32767
Clicks	No. of clicks in Dist	0 - 32767

Reply_set :

Operation to set up the user defined reply message which is sent upon completion of a motor move for time, clicks or distance. The motor number for which the reply message is to be used is specified in the 'reply_set' message. After receipt of this message the motors slot expects the next message that it receives to be the one used as a completion message. Therefore it is essential that the user sends the reply message to the motors slot immediately following the 'reply_set' message, in order to prevent other messages inadvertently being stored by the motors slot and used as reply messages for the specified motor. In order to clarify the receipt of a reply message 'verbose' reporting, if enabled, will output a message to the system window when the message arrives and is stored.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rs'
Motor#	Motor number	0 - 3

Reply_clear:

Operation to remove the user defined reply message from the slots memory. Now if a reply is requested upon completion of a move the slot will send the default message of 'M! motor#' to the reply slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rc'
Motor#	Motor number	0 - 3

Verbose:

This message switches the slot between verbose reporting mode and normal operation. Verbose reporting outputs informational messages in the system text window in order to keep the user more informed with the internal status of the slot. Operation of the slot is slowed down by enabling this option due to the outputting of text, so in time dependant applications it may be necessary to have verbose reporting disabled.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also:Servos slot.

4.4. Slot Name : SERVOS

Description: This slot provides operations to control analogue DC servo motors connected up to the output port on the interface. Facilities are provided to change the position and speed of the servo, and like the motor slot, reply messages can be requested to be sent upon completion of a servo move.

Init:

Operation to initialise a servo on a specified output line number.

Field Name	Meaning	Value range / Text
Command	Operation command	'In'
Servo#	Servo number	0 - 7
Line#	Output line number	0 - 4

Speed :

Operation to alter the speed at which a servo is moved from its current position to the newly requested one.

Field Name	Meaning	Value range / Text
Command	Operation command	'Sp'
Servo#	Servo number	0 - 7
Speed	Speed of servo	0 - 10

Position:

Operation to change the position of a servo. The full range of movement of the servo is mapped onto a numerical range of -100 to 100 where zero represents the mid-point of movement and -100 and 100 represent the extremities. If a reply slot is specified in the message then upon completion of the move either a user defined or default completion message will be sent to the requested slot. The default reply message format takes the form of the two characters 'S!' followed by the servo number. If no reply slot is specified then no reply will be sent.

Field Name	Meaning	Value range / Text
Command	Operation command	'Po'
Servo#	Servo number	0 - 7
Position	Position of servo	-100 - 100
Reply_to	Slot to reply to	Slot name

Reply_set :

Operation to set up a reply message which will be associated with a specific servo and sent to the specified slot when a servo manoeuvre has completed. The servo slot expects that immediately following this message it will be sent the message which is to be used as the reply message for this servo. It is therefore important to ensure that no other messages are sent to the servo slot between this 'reply_set' message and the message which is to be used for the reply. If 'verbose' reporting is enabled then the receipt of a reply message will be acknowledged by the slot in the system window. If a user defined message is not set up, reply messages can still be requested because the slot has a default reply message of 'S! servo#' which it will send upon completion of a move.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rs'
Servo#	Servo number	0 - 7

Reply_cancel:

Operation to remove the user defined message from the slot. If reply messages are requested on subsequent servo moves the default message will be sent.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rc'
Servo#	Servo number	0 - 7

Clear :

Operation to remove the specified servo from the output port, so freeing the output line for other uses.

Field Name	Meaning	Value range / Text
Command	Operation command	'Cs'
Servo#	Servo number	0 - 7

Clear_all :

Operation to remove all servos from the output port.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ca'

Verbose:

Operation to toggle between verbose reporting being on and off. When in operation the slot outputs informational text messages about the slot to the system window so keeping the user better informed about the internal status of the slot. Use of the verbose reporting option causes the slot to run slightly slower due to the text output, therefore in time dependant applications it may be necessary to have verbose reporting disabled.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Motors slot.

4.5. Slot Name: BUGGY

Description: This slot is designed to be used in conjunction with a two wheeled vehicle and provides operations to allow the control of two motors simultaneously in order to make control of the buggy much simpler. The slot provides operations to start and stop the buggy, run the motors for a pre-defined time period, number of clicks or distance. As well as running the motors in a straight line the facility is provided to perform measured turns in both clockwise and anticlockwise directions. If requested, reply messages can be sent upon completion of any one of the measured moves. The software is designed to drive a buggy where the drive wheels are at the centre of the chassis and all turns are axial.

Init:

Operation to initialise the buggy's motor numbers and output line numbers for control of the motors. Care must be taken to ensure that the line numbers on which the motors are initialised are numerically two apart because each motor occupies two output lines in order to achieve direction control. Because a motor requires two consecutive output lines line number six is the last one that can be allocated for motor control since number seven, the last output line, will be used for direction control.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mi'
L-motor#	Left motor number	0 - 3
L-output	Left motor line number	0 - 6
L-dirn	Left motor direction for forward motion	1 - Forwards -1 - Backwards
R-motor#	Right motor number	0 - 3
R-output	Right motor line number	0 - 6
R-dirn	Right motor direction for forward motion	1 - Forwards -1 - Backwards

Move:

Operation to move the buggy continuously in the specified direction. A stop message or another move message must be sent to the slot to cancel this move.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mv'
Speed	Speed of motors	0 - 4
Dir	Direction of movement	1 - Forwards -1 - Backwards

Move_for_time:

Operation to move the buggy in the specified direction and speed for a certain time period. Time is measured in tenths of seconds. If a reply slot is specified in the message then upon completion either a user defined message or the default completion message will be sent to the requested slot. The default reply message is the same as that used by the motors slot and takes the form 'M!', but without the motor number specified.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mt'
Speed	Speed of motors	0 - 4
Dir	Direction of movement	1 - Forwards -1 - Backwards
Time	Duration of move	0 - 32767
Reply_to	Slot to reply to	Slot name

Move_for_dist:

This operation moves the buggy in the specified direction and speed for the distance requested in the message. In order for this operation to function correctly the encoders for the motors must have already been initialised and calibrated. Consequently the units of distance used are dependant upon that calibration. If a slot is specified, a reply message will be sent upon completion of the move. The message will be a user defined one if one has been set up using the 'set_reply_msg' message, or else the default completion message of the form: 'M!' will be sent.

Field Name	Meaning	Value range / Text
Command	Operation command	'Md'
Speed	Speed of motors	0 - 4
Dir	Direction of movement	1 - Forwards -1 - Backwards
Distance	Distance for move	0 - 32767
Reply_to	Slot to reply to	Slot name

Move_for_clicks:

Operation to move the buggy at a specified speed and direction for a certain number of clicks from the encoders associated with each motor. Before this operation will function correctly the 'init_encoders' message must be sent in order to link an encoder with each wheel of the buggy. If a reply slot is specified in the message then upon completion of the move a reply message will be sent. This message will be either user defined, if one has been set up, or the default completion message of 'M!'.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mc'
Speed	Speed of motors	0 - 4
Dir	Direction of movement	1 - Forwards -1 - Backwards
Clicks	Number of clicks	0 - 32767
Reply_to	Slot to reply to	Slot name

Stop:

Operation to stop any movement of the buggy. If this message is sent while another move is in progress the move will be interrupted by the stop command.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ms'

Turn:

Operation to perform a continuous turn at a specified speed and direction until the motors are stopped with the 'stop' message, or until another message is sent to change the operation of the motors.

Field Name	Meaning	Value range / Text
Command	Operation command	'Tu'
Speed	Speed of motors	0 - 4
Dir	Direction of turn	1 - Clockwise -1 - Anticlockwise

Turn_for_time:

Operation to turn the buggy at a specified speed and direction for a pre-defined period of time. The units of time specified in the message are in tenths of seconds. A reply slot can be specified in the message and if this is done then upon completion of the move either a default or user-defined reply message will be sent.

Field Name	Meaning	Value range / Text
Command	Operation command	'Tt'
Speed	Speed of motors	0 - 4
Dir	Direction of turn	1 - Clockwise -1 - Anticlockwise
Time	Duration of turn	0 - 32767
Reply_to	Slot to reply to	Slot name

Turn_thru_angle:

Operation to turn the buggy at a specified speed and direction for a certain number of degrees. In order to calculate the number of degrees turned the encoders for each motor must have been previously initialised with the 'init_encoders' message and also calibrated with the 'calibrate_turn' message. Once again if a reply slot is specified either a user defined or default message of completion will be sent. The default completion message is 'M!'.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ta'
Speed	Speed of motors	0 - 4
Dir	Direction of turn	1 - Clockwise -1 - Anticlockwise
Angle	Angle of turn	0 - 32767
Reply_to	Slot to reply to	Slot name

Calibrate_turn:

Operation to specify the distance covered by the wheels in order to turn through the specified angle. For this calibration to operate correctly the encoders must have already been initialised and calibrated for distance with the 'calibrate_encods' message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Tc'
Angle	Calibration angle	0 - 360
Distance	Distance to cover this angle	0 - 32767

Init_encoders:

Operation to initialise two encoders to be associated with the two drive motors.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ei'
L-input#	Left encoder input line	0 - 7
R-input#	Rght encoder input line	0 - 7

Calibrate_encods:

Operation to calibrate the encoders by specifying what distance corresponds to how many clicks from the encoders. Each wheel is calibrated individually and consequently may be scaled differently giving interesting results to the behaviour of the buggy.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ec'
L-dist	Left encoder distance	0 - 32767
L-clicks	Clicks to cover dist	0 - 32767
R-dist	Right encoder distance	0 - 32767
R-clicks	Clicks to cover dist	0 - 32767

Set_reply_msg:

Operation to set up a user defined reply message for the buggy slot to send upon completion of any of the measured moves or turns. After receiving this message the slot will store the next message it receives as the reply message, so it is important to ensure that the slot does not receive any messages between the 'set_reply_msg' and the actual message to be used as a reply. If 'verbose' reporting is enable the slot will acknowledge in the system window the receipt and storing of a reply message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rs'

Clear_reply_msg:

Operation to remove the user defined reply message from the slot. Now if a reply message is requested upon completion of any of the measured moves or turns it will be the default message of 'M!' that will be sent to the requesting slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Rc'

Verbose:

Operation to toggle between verbose and non-verbose operation of the slot. When in verbose mode the slot outputs informational messages to the system text window which help to keep the user more fully informed as to the internal status of the slot. When in verbose mode operation of the slot is slowed down slightly due to the text output, therefore in time critical applications it may be necessary to run the slot in non-verbose mode to gain optimum performance.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Motor slot.

4.6. Slot Name : WANDER

Description: This slot generates random movements for driving a buggy. By default the slot communicates with the buggy slot which in turn drives the buggy through the motor slot. Before the wander slot is activated the buggy slot with which it is communicating must have been previously initialised. An example application for this slot might be to allow the buggy to wander randomly around the floor while monitoring two bump detectors at the front of the vehicle. When a collision occurs a sequence to perform an avoidance manoeuver could be triggered off and the point of collision could be plotted on the screen to build up a map of the room.

Init_wander:

Operation to initialise the wander slot. The initialisation consists of setting the speed of the buggy and specifying the maximum length of any single run and the maximum angle of any single turn. The units used to specify the lengths of run and turn are the same as those specified during initialisation of the buggy slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Wi'
Speed	Speed of motors	0 - 4
Max_run	Maximum length of run	0 - 32767
Max_turn	Maximum degree of turn	0 - 360

Message_dest:

Operation to set the slot to which the movement commands are sent. By default all messages are sent to the buggy slot, but if other slots exist in the system which can respond to the same messages as those specified for the buggy slot then the wander slot can be instructed to send them there by means of this message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Wm'
Dest_slt	Destination slot	Slot name

Stop:

Operation to stop all movement of the buggy.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ws'

Start :

Operation to start or restart movement of the buggy.

Field Name	Meaning	Value range / Text
Command	Operation command	'Wg'

Verbose:

Operation to toggle the slot in and out of verbose mode. This mode outputs informational messages to the system text window which keeps the user better informed as to the internal status of the slots. Operating in verbose mode may cause the slot to run more slowly due to the text output, therefore it may be necessary to operate the slot in non-verbose mode for time critical applications.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Buggy slot and Motor slot.

4.7. Slot Name : PLAYNAME

Description: This slot forms part of the sequencer and when instructed will play a sequence which is either in memory or on disk. It is important to ensure that the system clock is running before a sequence is played since playing back a sequence is a time dependant operation which will not work if the clock is not running.

Play_named_seq:

Operation to play the named sequence. Initially the slot will look for the sequence in memory and if it fails to find it there it will search in the current directory on disk. If the file remains still unfound an alert is flashed on the screen to indicate that fact.

Field Name	Meaning	Value range / Text
Seq_name	Name of sequence	15 chars max

See Also: Play slot.

4.8. Slot Name : PLAY

Description: This slot simply stops the sequencer from playing back any more messages in the sequence that it is currently playing. When a sequence is stopped if the sequencer stops sending any more of the recorded messages, however if a move of say motors or servos was in progress when the stop was requested the move itself will continue to completion.

Stop_play :

Operation to stop the current sequence being played.

Field Name	Meaning	Value range / Text
Command	Operation command	'stop'

See Also: Playname slot.

4.9. Slot Name : CRDREAD

Description: This application slot drives the card reader model. The model counts the holes in punched cards that are fed into it and tries to match the number of holes counted with the list of codes it has been told about. The cards have two rows of holes one is used as a set of timing holes and the other contains the actual code for the card. Two switches on the model monitor each row of holes. If a hole is encountered in the coded row of holes in conjunction with one of the timing holes then this is recorded as a coded hole. The timing holes in the card represent a six digit binary number. When a coded hole is encountered in the card the associated binary value of that hole is added to the code value for that card. The first timing hole is treated as the most significant bit of the six digit number. When the card reader has registered six timing holes it at-

Card_init :

Operation to initialise the card reader. This involves specifying which input lines will be used for the timing line and which will be used for the coded line of holes. The motor number and output control line for that motor are also specified in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ci'
Timing#	Input line for timing	0 - 7
Code#	Input line for coding	0 - 7
Motor#	Motor number	0 - 3
Line#	Output line for motor	0 - 6

Init_code :

Operation to initialise a card code. The code number to be matched is passed in the message, along with the slot name to which the reply message must be sent when the specified code is recognised.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ic'
Code#	Value of code	0 - 127
Reply_to	Slot to reply to	Slot name

Reader_start:

Operation to start the card reader off. The slot will wait for a card to be inserted in the machine and will then start up the motor to draw the card through.

Field Name	Meaning	Value range / Text
Command	Operation command	'Cs'

Reader_end :

Operation to stop the card reader slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ce'

Verbose:

Operation to toggle between verbose and non-verbose mode. In verbose operation informational text messages are output to the system text window to keep the user informed as to the status of the slot. The slot will run more slowly in verbose mode due to the text output, and might consequently miss some of the timing or coded holes causing the slot to fail to recognise certain punched cards.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: No other related slots.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: No other related slots.

4.10. Slot Name : IMAGES

Description: This slot provides facilities to load images up onto the screen and to manipulate them once they are there. Manipulation consists of moving them about the screen and making them visible or invisible.

Load_image :

Operation to load an image from disk into memory. The name of the image must be specified in the message and must not exceed eight characters excluding the file type suffix.

Field Name	Meaning	Value range / Text
Command	Operation command	'L'
Img_name	Name of image	8 char name

Hide_image :

Operation to hide an image that has already been loaded into memory. The image still remains in memory and can be moved about, however it is no longer visible on the screen.

Field Name	Meaning	Value range / Text
Command	Operation command	'H'
Img_name	Name of image	8 char name

Show_image :

Operation to perform the opposite of 'hide_image'.

Field Name	Meaning	Value range / Text
Command	Operation command	'S'
Img_name	Name of image	8 char name

Move_image :

Operation to move an image about the screen. All graphic screen areas are mapped onto a scale of 1000 in both x and y axes so these coordinates must be specified in the message to the slot. The coordinates passed in the message will be used to position the top left hand corner of the rectangle containing the image.

Field Name	Meaning	Value range / Text
Command	Operation command	'M'
Img_name	Name of image	8 char name
x	X screen position	0 - 1000
y	Y screen position	0 - 1000

Verbose:

Operation to toggle between verbose and non-verbose modes of operation. In verbose mode the slot outputs informational message to the system text window to inform the user as to the internal status of the slot. When in verbose mode the slot will run more slowly due to the text output, so for time dependant applications it may be necessary to run the slot in non-verbose mode.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: No relevant slots.

4.11. Slot Name : TURTLE

Description: This slot provides basic turtle graphics commands to allow easy drawing of lines on any one of the four graphics windows. Movement commands for the turtle consist of forward and backward moves, along with left and right turns. The slot also provides the facility to link up with another slot which is capable of driving a two wheeled buggy. Once linked up, this buggy slot will be sent messages like those sent to the buggy slot outlined in this document. This facility thus provides actual control of a real turtle while at the same time plotting its progress in one of the graphics windows. The coor dinate system for all of the graphics windows is (1000, 1000) irrespective of the size of the window. Changing the size of a window merely changes the area of the full window that is visible, it does not re-scale the window.

Move_home :

Operation to move the turtle to the centre of the window, i.e. (500, 500). If the pen is down then a line will be drawn from the current position to the centre.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mh'

Move_forwards:

Operation to move the turtle in the direction in which it is facing by the amount specified in the distance field.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mf'
Distance	Distance of move	0 - 1000

Move_backwards:

Operation to move the turtle in the opposite direction to that in which it is facing for the amount specified in the distance field.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mb'
Distance	Distance of move	0 - 1000

Turn_left :

Operation to turn the turtle left by the specified angle.

Field Name	Meaning	Value range / Text
Command	Operation command	'Tl'
By_angle	Angle to turn through	0 - 360

Turn_right :

Operation to turn the turtle right by the specified angle.

Field Name	Meaning	Value range / Text
Command	Operation command	'Tr'
By_angle	Angle to turn through	0 - 360

Pen_up :

Operation to lift the pen up and so effectively stop all drawing when any of the move or turn commands are used.

Field Name	Meaning	Value range / Text
Command	Operation command	'Pu'

n_down:

Operation to lower the pen and so start drawing when any of the turn or move commands are used.

Field Name	Meaning	Value range / Text
Command	Operation command	'Pd'

onnect_buggy:

Operation to connect the turtle slot up with a buggy driver slot. Once the connection has been made the turtle slot will send messages of the style outlined in the description of the buggy slot, thus controlling a two wheeled buggy in the same manner as the on-screen turtle. The default destination for the messages is the buggy slot, however if another slot exists in the system which can respond to buggy style messages then this destination slot can be altered by changing the slot field in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Bc'
Slot	Buggy control slot name	Slot name

rn_speed :

Operation to set up the speed at which turns will be performed by the buggy once it has been connected to the slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Bst'
Speed	Speed of turn	0 - 4

ve_speed :

Operation to set the speed at which straight runs will be performed by the buggy after connection to the turtle slot.

Field Name	Meaning	Value range / Text
Command	Operation command	'Bsm'
Speed	Speed of straight moves	0 - 4

t_window :

Operation to set the graphics window in which all the plotting from the turtle slot will be drawn. Any one of the four graphics windows may be used. The default window for turtle slot output is window one.

Field Name	Meaning	Value range / Text
Command	Operation command	'Ws'
Window	Window number	1 - 4

Verbose:

This option toggles the slot in and out of verbose reporting mode. When in this mode the slot outputs informational text messages to the system text window to inform the user as to the internal status of the slot. When in this mode the slot may run more slowly due to the increased text output, so for time dependant applications it may be necessary to run the slot in non-verbose mode.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Buggy slot.

4.12. Slot Name : GRAPHICS

Description: This slot provides basic drawing primitives to perform relative and absolute line drawings along with the plotting of points of different style. When plotting points the operation can be performed in several different modes, each of which affect the existing image on the screen and the plotted point differently, these differences are outlined in the 'plot' message below. Drawing and plotting can take place in any one of the four graphics windows and selection is made by the window field specified in each message received by the slot.

Clear :

Operation to clear the specified graphics window.

Field Name	Meaning	Value range / Text
Command	Operation command	'Clr'
Window	Graphics window number	1 - 4

Point :

Operation to plot a single pixel point in the specified window at the X and Y coordinates passed.

Field Name	Meaning	Value range / Text
Command	Operation command	'Poi'
Window	Graphics window number	1 - 4
X	X window coordinate	0 - 1000
Y	Y window coordinate	0 - 1000

Linerel:

Operation to draw a line relative to the current position. In the message are supplied the increments of X and Y which will be added to the current position in order to give the end point of the new line.

Field Name	Meaning	Value range / Text
Command	Operation command	'Lrl'
Window	Graphics window number	1 - 4
Dx	Increment in X coord	-32768 - 32767
Dy	Increment in Y coord	-32768 - 32767

Lineto :

Operation to draw a line from the current position to the absolute X and Y positio specified in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Lto'
Window	Graphics window number	1 - 4
X	X window coordinate	0 - 1000
Y	Y window coordinate	0 - 1000

Lineabs:

Operation to draw a line from the start X and Y coordinates passed, to the finish X and Y coordinates.

Field Name	Meaning	Value range / Text
Command	Operation command	'Lab'
Window	Graphics window number	1 - 4
X1	Line start X coord	0 - 1000
Y1	Line start Y coord	0 - 1000
X2	Line end X coord	0 - 1000
Y2	Line end Y coord	0 - 1000

Plot:

Operation to plot a specified style of point at the specified X and Y coordinates in a pre-defined drawing mode. The different drawing modes operate as follows: Replace mode plots the point on the screen no matter what is underneath it, so replacing the existing point with the one just plotted. XOR mode logically eXclusive-ORs the new point with the existing point on the screen. This option is useful when moving points around the screen because one XOR will plot in the reverse of the background colour and if followed by another in the same place then the point will be erased while leaving the original point as it was.

Field Name	Meaning	Value range / Text
Command	Operation command	'Plt'
Window	Graphics window number	1 - 4
X	X window coordinate	0 - 1000
Y	Y window coordinate	0 - 1000
Shape	Style of point to plot	0 = . 1 = 4x4 Pixel blob 2 = Ball
Rule	Mode of plotting	3 = REPLACE 6 = XOR

See Also: Images slot.

4.13. Slot Name : XYPLOT.

Description: This slot controls the xy-plotter model enabling the user to move about the paper and to draw lines. The plotter is a drum plotter, movement in the X axis is achieved by moving the pen assembly from left to right, and movement in the Y axis is achieved by rotating the drum clockwise or anticlockwise. Two motors are used in the model and both require encoders in order to measure the distances of the lines plotted. The pen is lifted up and down onto the paper by means of an electromagnet connected to one of the output lines of the interface.

Init_x_motor:

Operation to initialise the X axis control motor and define the paper width. The calibration of the encoder is achieved by specifying how many clicks represent a certain distance, so all subsequent distance calculations will be based on that relationship. The width of the paper is specified in the same units of distance used for calibrating the encoder and the width that is set is the effective plotting width. The effective origin of the paper should be set to one edge of the paper or the other and can be achieved by setting a positive or negative paper width. Which end of the paper the origin corresponds to depends on the polarity of the connection to the motors. The pen control line is simply an output line which is either held high or low to keep the pen on or off the paper.

Field Name	Meaning	Value range / Text
Command	Operation command	'Xi'
Motor#	Motor number	0 - 3
Mtr_line	Control line for motor	0 - 6
Enc_line	Input line for encoder	0 - 7
Clicks	No. of clicks in dist	0 - 32767
Dist	Distance	0 - 32767
Paper_x	Paper width	-32768 - 32767
Pen_line	Pen control line	0 - 7

Init_y_motor:

Operation to initialise the Y axis motor and encoders. Initialisation is exactly the same as that for the X axis, the only difference in the message being that no pen control line needs to be specified in this message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Yi'
Motor#	Motor number	0 - 3
Mtr_line	Output line for motor	0 - 6
Enc_line	Input line for encoder	0 - 7
Clicks	No. of clicks in dist	0 - 32767
Dist	Distance	0 - 32767
Paper_y	Paper length	-32768 - 32767

Move_by:

Operation to perform a relative move from the current pen position to the current position plus the increments in X and Y specified in the message. As the word 'move' suggests, no line is drawn with this operation.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mb'
Dx	Change in X coordinate	-32768 - 32767
Dy	Change in Y coordinate	-32768 - 32767

Draw_by:

Operation to draw a line from the current position to the current position plus the increments in X and Y specified in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Db'
Dx	Change in X coordinate	-32768 - 32767
Dy	Change in Y coordinate	-32768 - 32767

Move_to:

Operation to move the pen without drawing from the current position to the absolute position specified by the X and Y values passed in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Mt'
X_abs	Absolute X coordinate	-32768 - 32767
Y-abs	Absolute Y coordinate	-32768 - 32767

Draw_to:

Operation to draw a line from the current position to the absolute position specified by the X and Y values passed in the message.

Field Name	Meaning	Value range / Text
Command	Operation command	'Dt'
X_abs	Absolute X coordinate	-32768 - 32767
Y_abs	Absolute Y coordinate	-32768 - 32767

Verbose:

Operation to toggle the slot in and out of verbose reporting mode. When in verbose mode the slot outputs informational text messages to the system text window which keep the user informed as to the internal status of the slot. The slot will run more slowly with verbose reporting on, so in time dependant applications it may be necessary to run the slot in non-verbose mode.

Field Name	Meaning	Value range / Text
Command	Operation command	'V'

See Also: Motors slot.

4.14. Slot Name : PRINT-TEXT

Description: This slot provides the facility to output text to any of the text windows. Output can consist of single words with no carriage return or a line of text which is terminated by a carriage return and line feed.

Print_word :

Operation to output a single word to the specified text window.

Field Name	Meaning	Value range / Text
Command	Operation command	'W'
Window#	Text window number	0 - 4
Word	Text for output	Max of 16 chars

Newline:

Operation to output a carriage return and line feed to the specified text window.

Field Name	Meaning	Value range / Text
Command	Operation command	'N'
Window#	Text window number	0 - 4

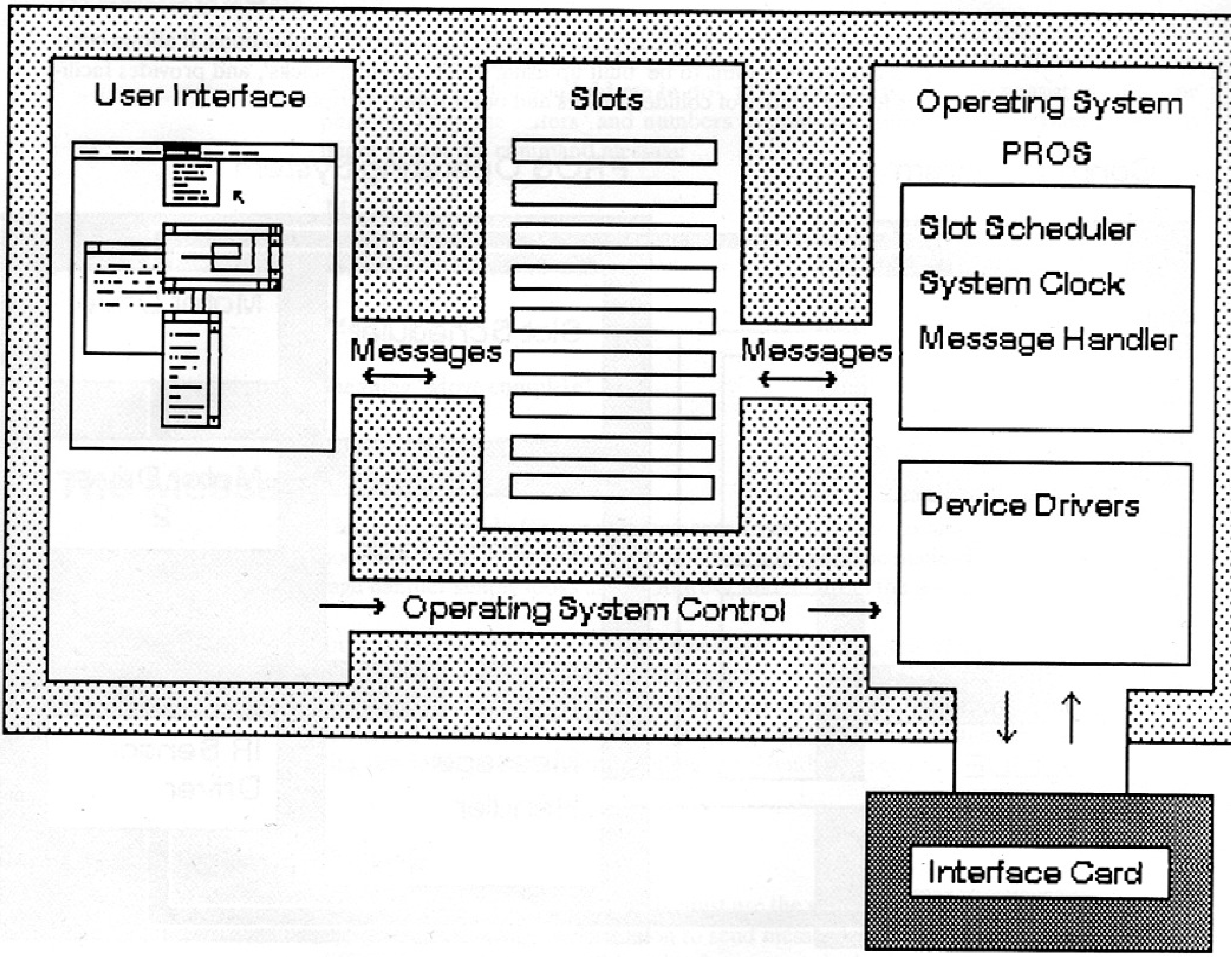
Print_line :

Operation to output a complete line to the specified text window, terminating the line with a carriage return and line feed.

Field Name	Meaning	Value range / Text
Command	Operation command	'L'
Window#	Text window number	0 - 4
Text	Text for output	Max of 16 chars

See Also: No relevant slots.

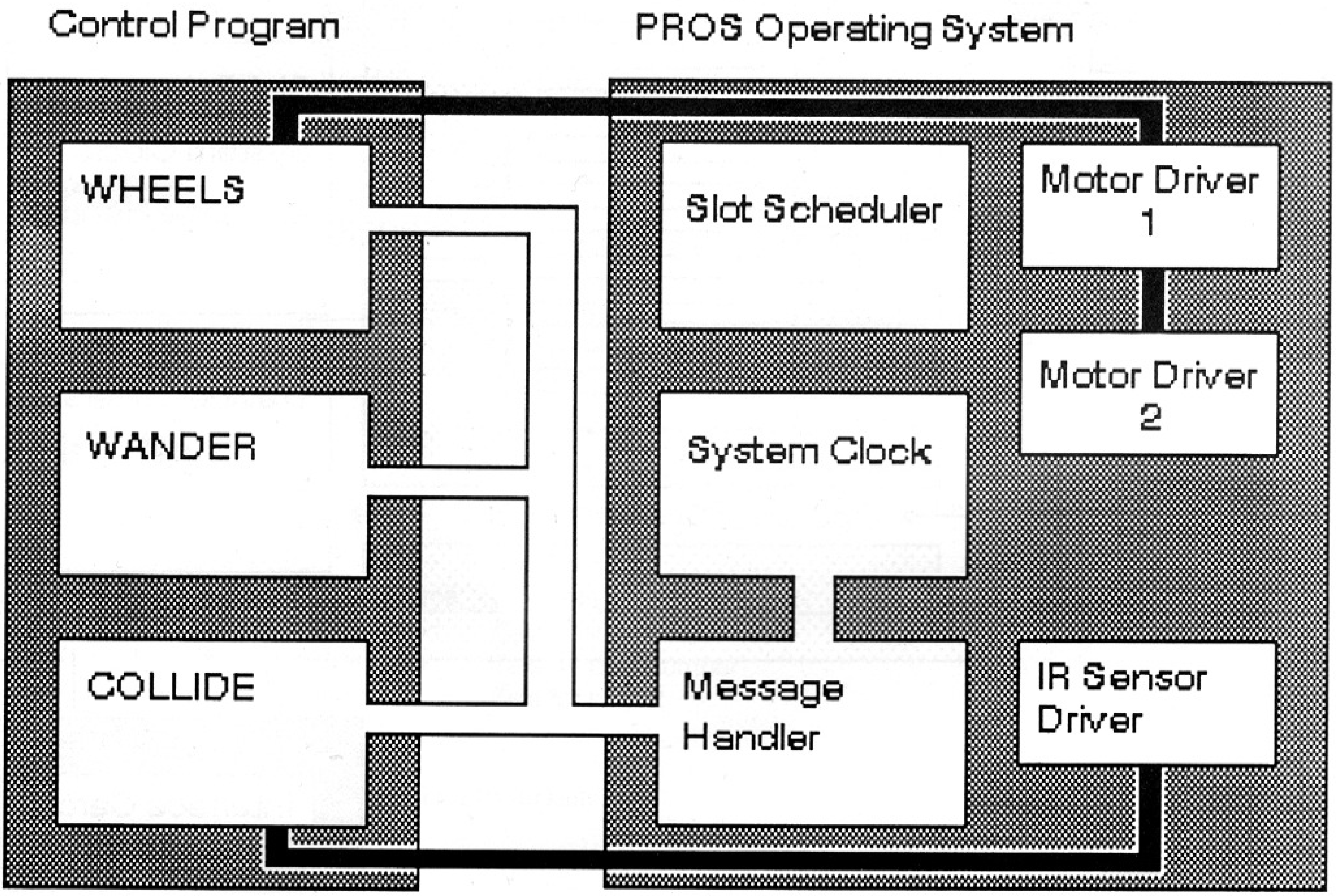
5. Software reference section


User Interface
Slots
Operating System
PROS
Slot Scheduler
System Clock
Message Handler
Messages
Messages
Device Drivers
Operating System Control
Interface Card

The structure of the ROBOKIT application is shown in above. It consists of the PROS Operating System including a number of device drivers, some slots, and the GEM-based user interface.

5.1. PROS Operating System, Slots and Drivers

PROS is an operating system for robots and other control applications. It allows complex control programs to be built up using small 'building blocks', and provides facilities for the control of collide, sensors and other robot 'peripherals'.


Control Program
PROS Operating System
WHEELS
WANDER
COLLIDE
Slot Scheduler
System Clock
Message Handler
Motor Driver 1
Motor Driver 2
IR Sensor Driver

The diagram above shows the structure of a simple robot control program.

Slots and the Slot Scheduler

The program is divided into a number of small units called slots. Each slot has responsibility for a particular aspect of the control of the robot. In the example, there is one slot for driving a pair of wheels, another for monitoring a collision detector, and another for deciding where the robot should go. The slot scheduler allows all the slots to run simultaneously; the wheels slot drives the wheels, and at the same time, the collide slot monitors sensors to see if the robot is about to hit something. It is obviously necessary for these slots to communicate with one another. When collide detects an obstacle in the path of the robot, it must tell wheels to stop before the obstacle is hit.

Messages

Slots communicate by sending messages. Most messages contain commands; one slot tells another slot to do something. Some messages, however, contain information; "I've finished doing what you told me to do". Messages do not usually consist of words or phrases. They use letters and numbers to convey information. For example, wheels might accept the command message

Mf 20

meaning "Move forward 20cm". When it has finished, it might send back the reply

Mc

meaning "Move complete".

The Message Handler

This is responsible for passing messages from one slot to another. It is rather like a postman. Each message has a destination 'address' (the name of a slot), and the message handler simply looks at this address, and delivers the message.

In our example, then, wander sends messages to wheels, telling it to move forward, turn, stop, and so on. Whenever wheels sends a reply message to wander to indicate that a command has been completed, wander chooses something else to do, and sends a new command to wheels. While all this is going on, collide must watch for obstacles by asking the infrared sensor for a reading every tenth of a second.

The System Clock

To time a tenth of a second, collide must use the system clock. This is part of the operating system, and can be called upon to send messages to slots at whatever intervals the different slots require. Collide asks the system clock for messages every tenth of a second. Each time it receives a system clock message, collide reads the infrared sensor, and sends an 'emergency stop' message to motors if an obstacle has been detected.

Device Drivers

The Operating System also contains device drivers. In our example there are three; One controls the infrared sensor, while the other two drive the motors. Device drivers are similar to slots, in that they all run simultaneously.

The User Interface

The User Interface uses GEM to provide a 'front end' to the Slots and Operating System. Menus and forms are used to send messages to slots, and slots can print text or draw graphics in windows. The behaviour of the Operating System can also be controlled, by slowing down the System Clock, or stopping it altogether. The Message Trapping system allows you to intercept messages as they pass through the Message Handler. Using images and icons on the backdrop (desktop), control panels can be constructed to allow easy real-time control of your robot using the mouse.

These facilities are described in detail in the manual. Spend some time working through the examples to get used to the 'feel' of the user interface.

The ROBOKIT Program

The slots and device drivers mentioned above are just examples. Your ROBOKIT application contains many slots, and they are described in the Slot Reference secion of the manual. The application also contains device drivers for motors, servos, shaft encoders, inputs and outputs. You can't send messages to the device drivers themselves, but they are all accessed through slots.

5.2. File system structures

These notes apply to Atari ROBOKIT version 1.65 and later versions.

ROBOKIT.PRG expects to find the following files in the current directory when it is loaded:

ROBOKIT.RSC

PROCESS.ADR

MINIT.MTX

MRUNT.MTX

various.MSG

where "various" indicates several files of the same type.

These files must be in the current direcory, and cannot be moved elsewhere.

This doesn't mean that you have to have ROBOKIT.PRG in the root directory. It could be in a directory called ROBOKIT, for example.

From version 1.65, sequences and backdrop images and setup (.RSU) files may be located in directories other than the current one. For example, you may wish to create a SEQS directory to hold sequences. When you "save setup" from ROBOKIT, the FULL PATH NAME of each sequence and image loaded is stored in the setup file. Earlier versions of ROBOKIT saved only the filename, and expected all files to be in the current directory.

When restoring a setup, ROBOKIT first tries to find the file using its full path name, then looks in the current directory.

5.3. MAKEDRP.PRG

The Makedrp utility is used to prepare images for loading onto the ROBOKIT backdrop. It accepts various formats of file, and converts them to the ROBOKIT .DRP format. Makedrp is more than just a file conversion utility. It allows images to be clipped, inverted and scaled, in order to make them an appropriate size for the particular backdrop being designed.

How to use MAKEDRP

Double click on MAKEDRP.PRG to load the program. You will be asked to select the type of file you wish to load. Click on a file type, and you will see a file selector for the appropriate type of file. Select the file in the usual way.

Once the file has been loaded, it will appear in a window. You can now perform three different operations on the image, using the options from the operations menu.

Clip

This clips the image (reduces its area) to the part of the image which is actually visible in the window. To align the image for this operation, resize the window and then scroll the image in the window using the scroll bars.

Scale

This scales the image down from its full size to an area equal to the size of the window. For example, if the top half of the image is visible in the window when the scale operation is selected, the whole image will be 'squeezed' vertically so that it fits into the window. The image will be half its original height.

Note that you can only scale down. You can't use the scale operation to enlarge an image.

vert

This inverts the colours of the image. Black becomes white and white becomes black.

ndo

This restores the image to the state it was in before the previous clip scale invert or undo operation.

When you have finished operating on the image, save it to a .DRP file using the save option from the file menu. This is the file you will load onto the ROBOKIT backdrop.

otes

ROBOKIT uses monochrome (black and white) images only. You can load colour images into MAKEDRP, but anything non-white will be regarded as black. This colour mapping does not take the pallette settings into account either. Sensible results can be obtained from NEOChrome or DEGAS by using one colour on a white background with the default palette settings.

You can only load one image at a time into MAKEDRP.

4. Accessing the ROBOKIT interface card

The ROBOKIT interface card has been designed to operate in conjunction with the ATARI ROBOKIT software. If you wish to access it from another system, the following notes may be of use to you. They are fairly technical in nature, and describe the use of assembly code to access the board. An example of accessing the card from ST BASIC is also given. Other BASIC systems may use a slightly different syntax.

The ROBOKIT interface card is connected to the ST's ROM cartridge port, and is mapped into (the whole of) the ROM-3 address space. To access the card you will need to write a rather tricky little piece of code which is given here is 68000 assembler. The problem with the cartridge port is that you can't write to it (you get cherry-bombs if you try). We fool the system by doing read operations and using the address as the data to output. So whenever we read from anywhere in the ROM3 space, the address we read from (actually bits 1 to 8 of the address bus) are sent to the outputs on the interface card, and the inputs are read as normal data. The code fragment below uses d0 to hold the output data. A0 points to ROM3, so we do an indexed read using (a0,d0) to output the data and read the inputs. The 68000 doesn't have a bit zero (odd) address line, so we shift the data left one bit before using it to put the output data onto address bus bits 1 to 8.

```
ROM3  equ         $fa0000 ; start of ROM space
moveq.l         #0,d0   ; clear all of D0
move.w  < output data > ,d0     ; get stored data for
output
lsl.l   #1,d0   ; shift into bits 1..8
lea     ROM3,a0 ; use A0 as base
move.w  0(a0,d0.l),d0   ; for indexed read.

* The above instruction MUST use d0.l (rather than .w) to avoid getting
* sign extension into the upper word.Hence the first instruction, which
* clears d0.l to ensure that the upper word is zero.
```

The card can be accessed by using a PEEK instruction from BASIC:

```
10 ROM3% = &HFA0000
     20 OUT% = 0
     30 IN% = 0
     100 RDWR:
     101 REM Subroutine to access card.Sends OUT% to outputs and 102 REM
places input data in IN%
     110 IN% = PEEK(ROM3% + (OUT%*2))
     120 RETURN
```

So to switch outputs 0 and 1 ON:

```
OUT% = 3 : GOSUB RDWR
```

To read the inputs:

```
GOSUB RDWR : PRINT IN%
```

The inputs on the card are normally held high by 'pull-up' resistors. This means that you will get a reading of all ones (FF hex) with nothing connected to the card. The inputs are designed to be triggered by connection to zero volts (the '0v' terminal) through a device such as a microswitch. The LEGO optical encoder bricks, which may be connected to inputs 0 and 1, will give a '1' when they sense 'dark' and a '0' when they sense 'light'.

The ROBOKIT software actually inverts the input data when it is read, so that a closed switch (0v input) is represented by a '1' and an open switch (5v input) is represented by a '0'.This also has the effect of making the LEGO encoder give a '1' for 'light' and a '0' for 'dark', which is rather more intuitive than before. The inversion can be performed in assembler by addition of

```
not.w   d0
```

after the code fragment given above. From BASIC, substitute line 110 in the example with the following:

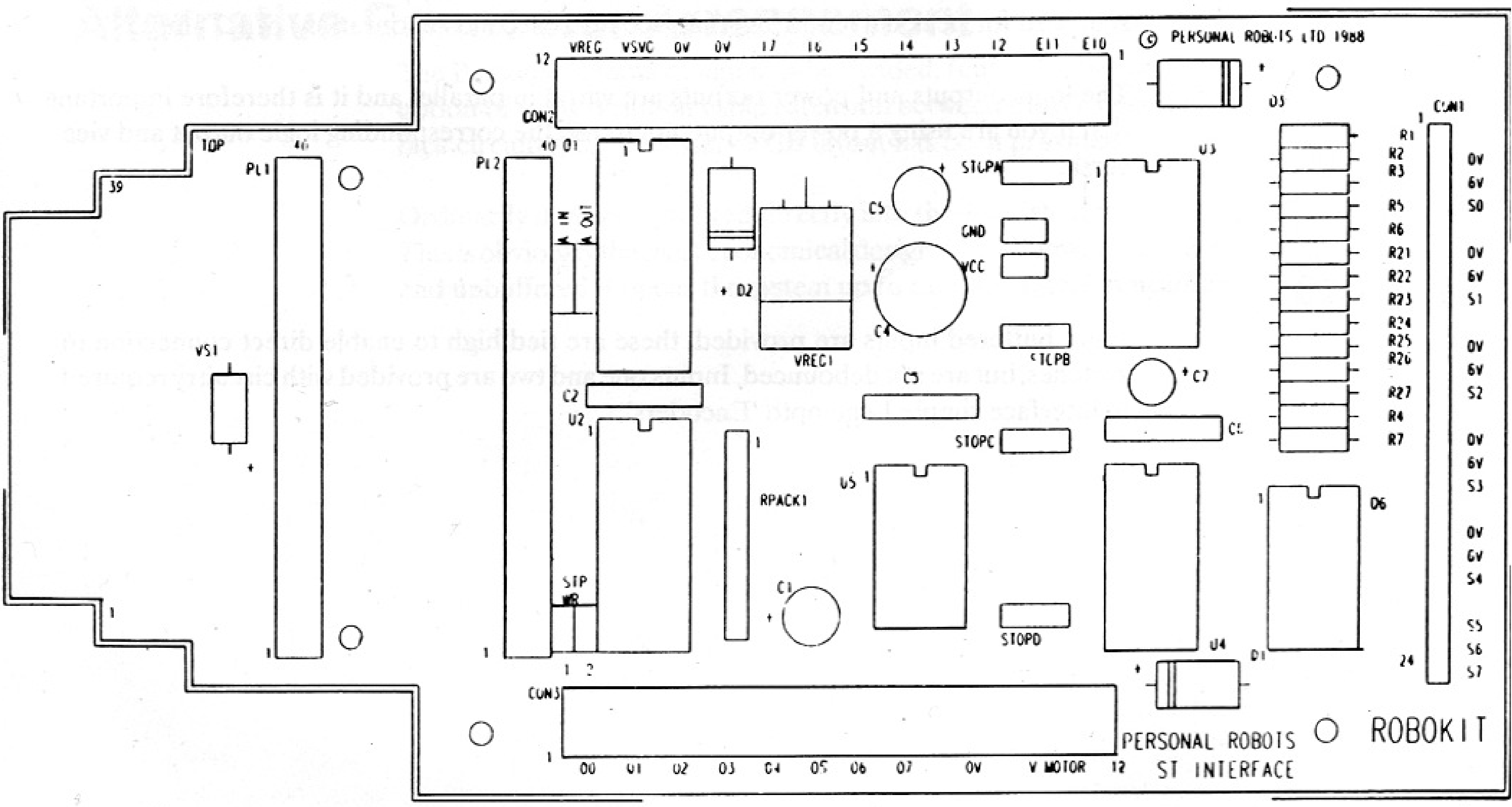
```
110 IN% = 65535 - PEEK(ROM3% + (OUT%*2))
```

6. Hardware Interface Reference Section

The Robokit expansion board is designed to provide basic I/O facilities for the ST. These facilities include an interface to motors, servos, solenoids, switches and Lego optical shaft encoders, as would be used in simple robotic applications. The board is self-contained and draws its power from the Atari ST, so the only additional requirement is for an external power supply for the motors themselves.

- A bread boarding area is provided for prototyping additional circuitry.
- Sixteen channels (lines) of I/O are provided, eight for input and eight for output.

The PCB layout


© PERSONAL ROBOTS LTD 1988
VREG
VSVC
0V
0V
I7
I6
I5
I4
I3
I2
EI1
EI0
12
1
CON2
TOP
40
39
PL1
40 01
PL2
A IN
A OUT
+ D2
VS1
+
C2
U2
1
STP
WR
1 2
CON3
00
01
02
03
04
05
06
07
0V
V MOTOR
RPACK1
C1
C5
C4
VREC1
GND
VCC
STOPA
STOPB
STOPC
STOPD
C5
U5
U3
C7
C6
U4
D1
D5
U6
CON1
R1
R2
R3
R5
R6
R21
R22
R23
R24
R25
R26
R27
R4
R7
0V
6V
S0
S1
S2
S3
S4
S5
S6
S7
24
PERSONAL ROBOTS
ST INTERFACE
ROBOKIT

6.1. Board functions

Outputs

Of the eight outputs each may be used as a logic or power output. A single logic output will drive a Futaba DC servo, whilst a single power output will drive a light bulb, solenoid, or a DC motor - in one direction only. Two power outputs are required to drive a DC motor bi-directionally, so that the interface could drive a maximum of four bi-directional motors. However, the channels may be freely mixed to provide combinations such as the following:

Two servos	2 channels.
One solenoid driver,	1 channel.
Two bi-directional DC motors	4 channels.
One unidirectional DC motor	1 channel.
Total:	**8 channels.**

The power outputs are rated at 600 milliamps (0.6 amps) per channel continuous current, and for peak currents (less than 100 microseconds in duration) of 1.2 amps.

The logic outputs and power outputs are wired in parallel and it is therefore important that if you are using a power output, not to use the corresponding logic output and vice versa.

Inputs

Eight buffered inputs are provided, these are tied high to enable direct connection to switches, but are not debounced. Inputs one and two are provided with circuitry required to interface simple Lego opto 'Encoders'.

6.2. Options

The following options have been allowed for on the PCB.

Power supplies

ST or Off-board.

When the interface is powered through the ST (as is the case for the projects shown in this manual)then a high voltage suppresser VS1 is included in order to protect the ST, against high voltage transients, however no protection is given against a short circuit on the ST supply produced by the user.

Alternative Connector Arrangement

The Personal Robots design may be divided, (cut), into two sections. This allows for the option of using a ribbon cable extension between the ST edge connector and the interface circuitry, the necessary PCB layout has been provided.

Ordinarily it may be plugged directly into the ST,with consequent restriction in access. This is obviously the most economical design option, however as the bussing is unshielded and unbuffered it opens the system up to external interference/noise.

6.3. Power up sequence

The following practices should be observed:

1) With the ST and interface boards both off, insert the interface board into the ST ROM cartridge port.

2) If the interface board has been modified to allow the use of an alternative power supply to that of the ST, the interface board power should be switched on before the ST.

3) Switch on the ST.

4) If any problems are observed at this stage then all connected power supplies should be switched off and disconnected as quickly as possible. All connections must be checked carefully and the problem identified and rectified before switching on again.

6.4. Driving motors

Two Motor supply rails are provided VM, (Ordinary DC motor drive), and VS, (Futaba type DC Servo motor drive). These supply lines may be used in common such that VM = VS = six volts, but for higher power motor drives the VM supplies will be separated giving VM an upper limit of 36 volts, whilst VS remains at six volts.

Servo use

The connector is designed to allow the direct connection of up to five Futaba Radio Control Model servo motors. These require three connections: 0 volts, 6 volts, and the PWM logic control signal (0 - 5 volts), these are all clearly identified on the PCB, with the logic control signal being labeled Sn, where n is the number of the servo connector.

Each Servo requires the use of one output channel, and will mean that the associated output line eg O5 for S5 can not be used.

An additional three servos may be driven from the board, however the use of these requires a suitable connector to take the control signals from the board and add the necessary 6 volt and 0 volt wires.

Servo power supply Unlike the use of DC motors which are powered by 'V Mtr' through the output buffers: O0 - O7, the servo power is taken directly from the off board 'V svo' supply, which normally will be 6 volts, however its current rating is only limited by that which can be safely handled on the PCB, and in all but the most unusual circumstances this will therefore be limited by that of the power supply or batteries which are used to provide 'V svo'.

DC motor use

The output channels are provided by two chips, such that the outputs may be grouped into two nibbles, (groups of four). A total of five Watts, eg 4 x 1.25 Watt motors/solenoids etc., may be driven from each chip. This means that this total can be divided in any way between the channels in a particular group ie O0 - O3, and O4 - O7. (Note that 1.25 Watts = 6 volt motor at 200 mA, or a 3 volt motor at 400 mA). All output channels have internal protection diodes.

Bi-directional motor drive

Each Bi-directional motor drive requires the use of TWO consecutive output channels, eg O0 & O1, or O2 & O3 etc. The motor should be connected between the two outputs.

Uni-directional motor and solenoid/relay drive

EachUni-directional motor drive requires the use of one output channel, eg O0, O1, O2 etc. The motor should be connected between the output channel and ground. All of the buffered outputs can drive a solenoid or other component requiring high current drive such as relays, the limit on all these outputs is the same as that for driving motors.

6.5. The Prototyping Area

This is suitable for prototyping simple circuitry which can then be accessed through the interface. The area consists of strip type tracking which can be cut as required by means of a readily obtainable board cutter, (as available through most electronic suppliers).

The Prototyping Area is not available on the current version of ROBOKIT

Soldered connection points

All the buffered motor output lines O0 to O7 and the input lines EI0, EI1 and I2 to I7 are taken to the prototyping area to allow wires to be directly connected. Supply lines These are all taken onto the prototyping area for ease of direct access. The labels on the PCB have the following meanings:

Vcc (or 5 Volts)

If the ST supply is in use then this is the logic supply which is taken directly from the ST, as such any use of this in user prototype circuits is not advised.

An alternative is provided where the addition of a few components on board allows the use of an independent unregulated supply, which can then be regulated on board to provide approximately 1.5 Amps at 5 volts.

IF AN ALTERNATIVE SUPPLY IS USED THE ST Vcc/5 voltSUPPLY MUST BE DISCONNECTED BEFORE THE ADDITION OF THE ON BOARD REGULATOR ETC..

0V (or Ground/'negative') line

This is commonly called the ground, common or zero volts line, it is not a negative supply line although it may be used to connect the 'negative' wire from a motor.

The Prototyping Area is not available on the current version of ROBOKIT

V unreg, (unregulated input line

This may be used to provide the optional on board regulator with a supply.

IF AN ALTERNATIVE SUPPLY IS USED THE ST Vcc/5 volt SUPPLY MUST BE DISCONNECTED BEFORE THE ADDITION OF THE ON BOARD REGULATOR ETC..

The regulator can accept inputs between 7.3 and 35 volts, although approximately 10 volts is recommended. The LM34O-5 can provide at most 1.5 Amps.

V Mtr, Motor voltage, (Off board motor supply)

This can be unregulated, eg supplied from a battery or other DC supply. It should be compatible with the motors in use, and should not exceed 36 Volts.

Although the output buffers can accept this high voltage, 12 volts is a recommended maximum, with 3 to 6 volts motors being most suitable.

V Svp, Servo supply voltage, (Off board supply)

This supplies the servo motors, (Futaba RC servos, or other 100% compatibles). This can be unregulated, eg supplied from a battery or other DC supply, it should be 6 volts.

IF AND ONLY IF THE V MTR SUPPLY IS 6 VOLTS, THESE TWO MAY BE CONNECTED TOGETHER, AND RUN FROM THE ONE SUPPLY.

6.6. WARNINGS & ABSOLUTE MAXIMUM RATINGS

DANGER : DO NOT CONNECT MAINS OR OTHER HIGH VOLTAGES TO THE BOARD. IT IS DESIGNED TO TAKE A MAXIMUM VOLTAGE OF 36 VOLTS ONLY

Additional warnings!!!

1)There is no reverse motor supply protection, the application of such voltages will cause potential damage to all connected circuits.

2)Do observe the restrictions on the motor supplies.

3)It is not recommended that you use the ST 5 volt/Vcc supply to power user prototyped circuits, the onboard regulator should be fitted and the ST supply disconnected before use.

Output drive buffers:

1) 6OO milli Amps / Channel.

2) 1.2 Amps peak output current for 100uS.

3) Automatic over temperature protection. Total package power dissipation 5 Watts. (at T ground pins = 80 degrees centigrade).

4) V mtr, input motor supply voltage up to 36 volts.

5) V Svo, input 6 volts max.

6) Vcc, output 5 volts, (Current limited by ST or 1.5 Amps if regulator fitted and external supply used, external supply recommended if the prototype area to be used).

7) V unreg, input 35 volts max.

6.7. User upgrades and options

Power supply upgrade

Addition of on-board regulator

i) Please consult a drawing of the board.

ii) BOTH links STP WR 1 and STP WR 2 MUST BE BROKEN. These are situated by U2 near pin 10.

iii) The following components are required:

C4 = 220uF minimum 6 volts electrolytic capacitor.
C5 = 0.22uF minimum 6 volts electrolytic capacitor.
C6 = 0.1uF ceramic type.
VREG1 = LM340 - 5 volt regulator.

Component polarities: (looking at the board with the ROBOKIT legend at the top).The voltage regulator has pin one identified by the proximity of the legend which is next to it. Both C4 and C5 have the + pin in line with pin 1 of the regulator, that is to their left, towards U1 the 74LS374.

iv) These components should be soldered onto the board, having observed the correct polarities.

v) A check of the correct assembly and soldering of these components should be made.
vi) Assuming that the check is satisfactory and that the links are both broken, an off board unregulated supply can be connected and used, (ie 'V unreg').

vii) The use of such a power supply requires that the interface board is switched on before the ST.

Ribbon Cable Connector to the ST

The PCB is designed in such a manner that it can be cut in two allowing connection to an ST via a ribbon cable and IDC or transition type connectors. This means that the board does not have to be placed next to the ST. A maximum length of 45 cm/ 18 inches is recommended.

Further Expansion via Ribbon Cable

Alternatively the board can be used as supplied, (ie in one piece), and an IDC/transition connector fitted near to the edge connector site and used to connect an additional circuit/prototyping board.

Two Robokit boards per ST!!!!

The use of the IDC site to allow connection to additional boards means that two ROBOKIT INTERFACE boards can be connected to the ST, however if this is done, the ADDRIN and ADDROUT links must be altered accordingly. (If these are not altered the boards may be damaged in use due to contention).

Output buffer heat sinking

The only heat sinking is provided on the PCB by extended ground areas in connection with the motor drive buffers. Additional heat sinking will extend the output drive capability, and can be provided by 'DIL', dual-in-line IC mounted heat sinks, which should be placed on U3 and U4.

Motor control

An optional gate package (74LS32), has been included which allows further flexibility in the use of the motor drive chips. Although this is not detailed here.

PCB Link Options

The following links are provided on the board:

1) ST PWR1 labeled on the board as 'STP WR1'.

2) ST PWR2 labeled on the board as 'STP WR2'.

3) ADDRIN This allows the selection of the input port to be either NROM3 or NROM4.

4) ADDROUT This allows the selection of the output port to be either NROM3 or NROM4.

5) STOP A labeled 'STOPA' on the PCB.

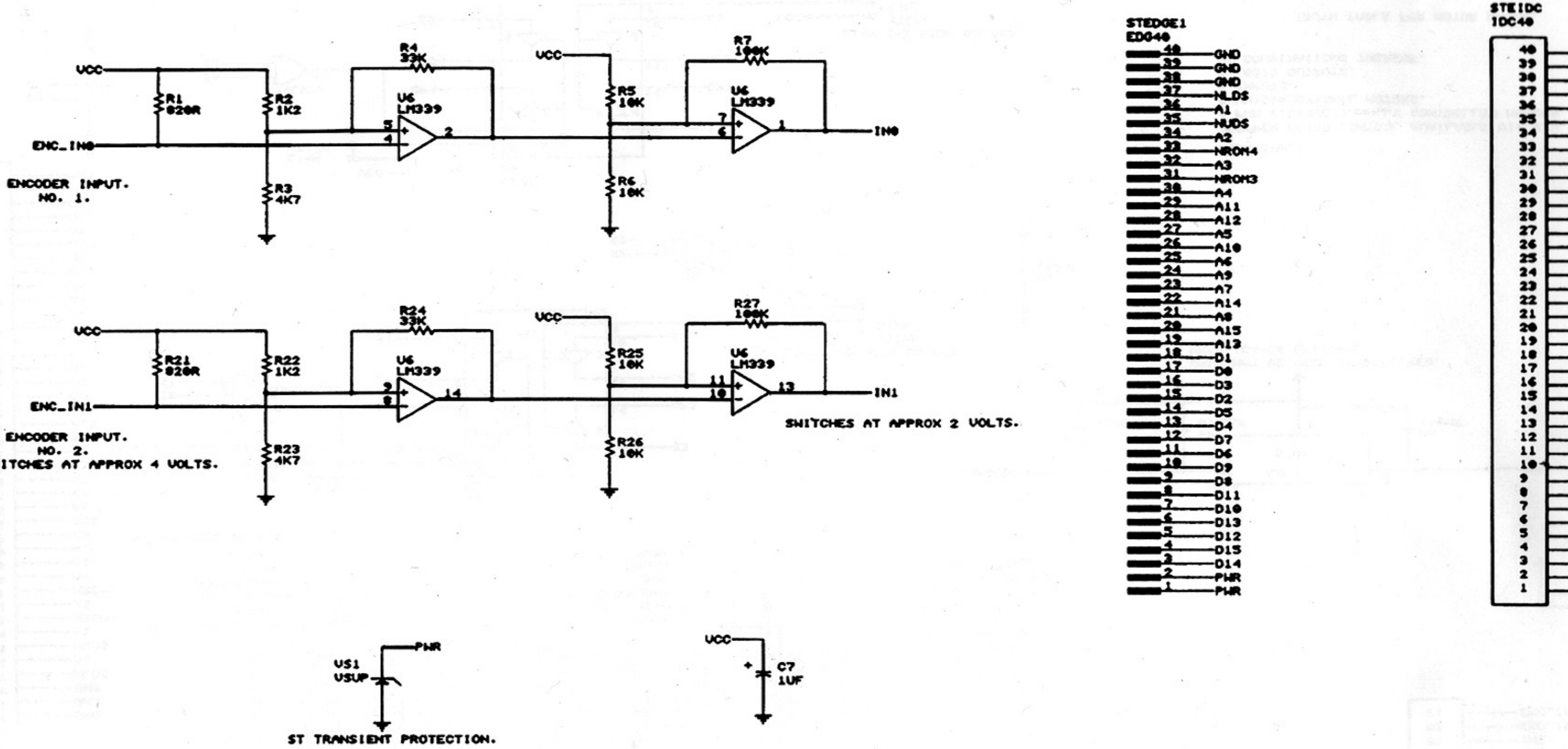
6) STOP B labeled 'STOPB' on the PCB.

7) STOP C labeled 'STOPC' on the PCB.

8) STOP D labeled 'D' on the PCB.

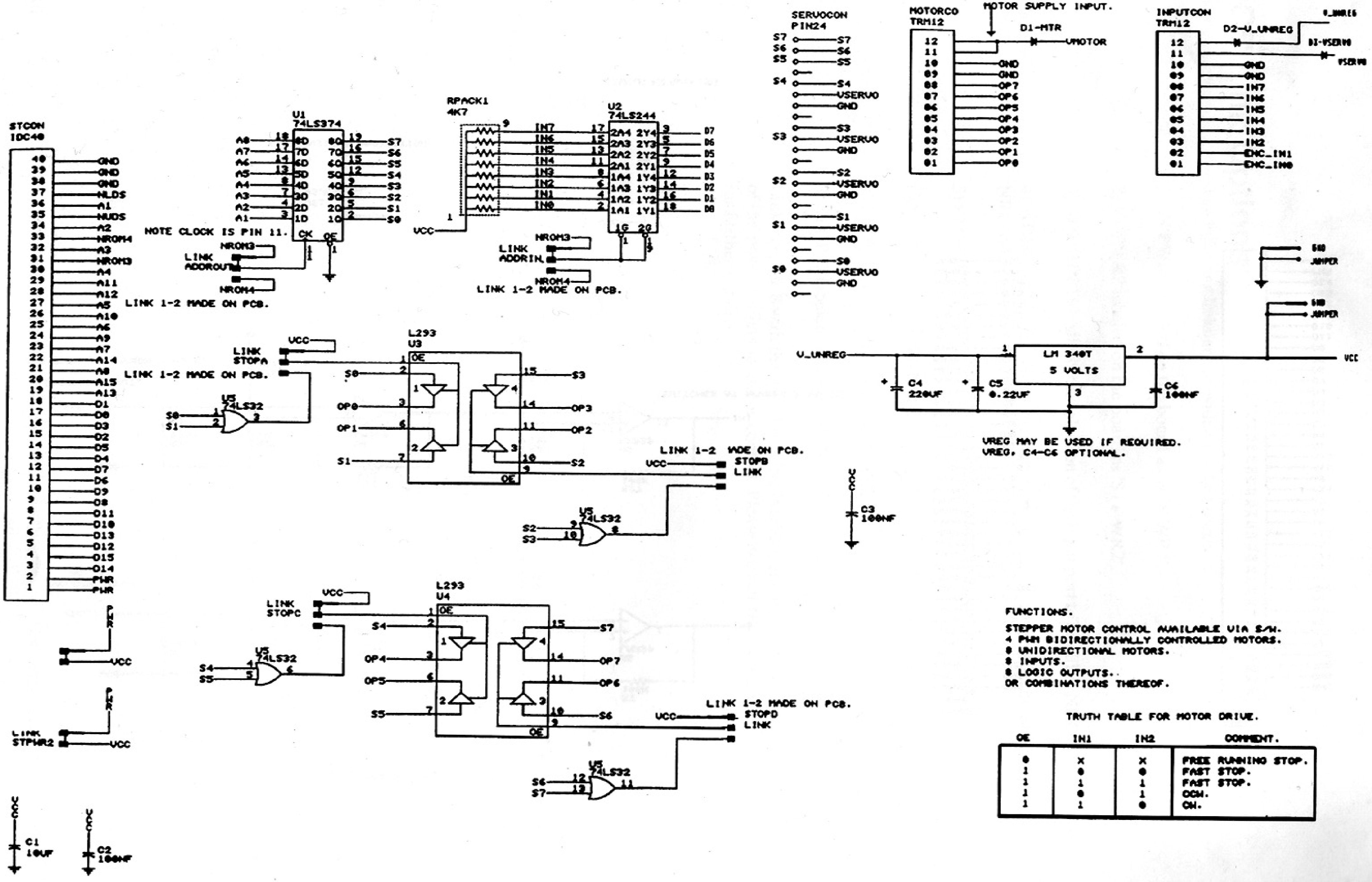
Links 5,6,7 & 8 with the use of IC U5 (74LS32), allow a subtle difference in the control of the power buffers in motor control. Please consult the relevant manufacturers data for further details.

ENCODER INTERFACE.


VCC
R1
820R
R2
1K2
R3
4K7
ENC_IN0
ENCODER INPUT.
NO. 1.
R4
33K
U6
LM339
5
4
2
VCC
R5
10K
R6
10K
R7
100K
U6
LM339
7
6
1
IN0
VCC
R21
820R
R22
1K2
R23
4K7
ENC_IN1
ENCODER INPUT.
NO. 2.
SWITCHES AT APPROX 4 VOLTS.
R24
33K
U6
LM339
9
8
14
VCC
R25
10K
R26
10K
R27
100K
U6
LM339
11
10
13
IN1
SWITCHES AT APPROX 2 VOLTS.
VS1
VSUP
PWR
ST TRANSIENT PROTECTION.
VCC
C7
1UF
STEDGE1
ED040
40 GND
39 GND
38 GND
37 NLDS
36 A1
35 NUDS
34 A2
33 NROM4
32 A3
31 NROM3
30 A4
29 A11
28 A12
27 A5
26 A10
25 A6
24 A9
23 A7
22 A14
21 A8
20 A15
19 A13
18 D1
17 D0
16 D3
15 D2
14 D5
13 D4
12 D7
11 D6
10 D9
9 D8
8 D11
7 D10
6 D13
5 D12
4 D15
3 D14
2 PWR
1 PWR
STEIDC
IDC40
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1

ROBOKIT PCB March 1989

ROBOKIT PCB
MARCH 1989


STCON
IDC40
GND
GND
GND
NLDS
A1
NUDS
A2
NROM4
A3
NROM3
A4
A11
A12
A5
A10
A6
A9
A7
A14
A8
A15
A13
D1
D0
D3
D2
D5
D4
D7
D6
D9
D8
D11
D10
D13
D12
D15
D14
PWR
PWR
LINK
STPWR2
VCC
C1
10UF
C2
100NF
U1
74LS374
A0
A1
A2
A3
A4
A5
A6
A7
S0
S1
S2
S3
S4
S5
S6
S7
CK
OE
LINK
ADDROUT
NROM3
NROM4
RPACK1
4K7
VCC
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
U2
74LS244
1G
2G
D0
D1
D2
D3
D4
D5
D6
D7
LINK
ADDRIN
NROM3
NROM4
L293
U3
OE
S0
OP0
OP1
S1
S3
OP3
OP2
S2
LINK
STOPA
VCC
U5
74LS32
S0
S1
STOPB
LINK
VCC
U5
74LS32
S2
S3
L293
U4
OE
S4
OP4
OP5
S5
S7
OP7
OP6
S6
LINK
STOPC
VCC
U5
74LS32
S4
S5
STOPD
LINK
VCC
U5
74LS32
S6
S7
SERVOCON
PIN24
S7
S6
S5
S4
USERVO
GND
S3
USERVO
GND
S2
USERVO
GND
S1
USERVO
GND
S0
USERVO
GND
MOTORCO
TRM12
MOTOR SUPPLY INPUT.
D1-MTR
VMOTOR
GND
GND
OP7
OP6
OP5
OP4
OP3
OP2
OP1
OP0
INPUTCON
TRM12
D2-V_UNREG
V_UNREG
D1-VSERVO
VSERVO
GND
GND
IN7
IN6
IN5
IN4
IN3
IN2
ENC_IN1
ENC_IN0
GND
JUMPER
GND
JUMPER
V_UNREG
LM 340T
5 VOLTS
VCC
C4
220UF
C5
0.22UF
C6
100NF
C3
100NF
VCC

NOTE CLOCK IS PIN 11.

LINK 1-2 MADE ON PCB.

VREG MAY BE USED IF REQUIRED.
VREG, C4-C6 OPTIONAL.

FUNCTIONS.

STEPPER MOTOR CONTROL AVAILABLE VIA S/W.
4 PWM BIDIRECTIONALLY CONTROLLED MOTORS.
8 UNIDIRECTIONAL MOTORS.
8 INPUTS.
8 LOGIC OUTPUTS..
OR COMBINATIONS THEREOF.

TRUTH TABLE FOR MOTOR DRIVE.

OE	IN1	IN2	COMMENT.
0	X	X	FREE RUNNING STOP.
1	0	0	FAST STOP.
1	1	1	FAST STOP.
1	0	1	CCW.
1	1	0	CW.